Mixing Confidential Transactions: Comprehensive Transaction Privacy for Bitcoin

Tim Ruffing[1] and Pedro Moreno-Sanchez[2]

[1]Saarland University
tim.ruffing@mmci.uni-saarland.de
[2]Purdue University
pmorenos@purdue.edu

Abstract. The public nature of the blockchain has been shown to be a severe threat for the privacy of Bitcoin users. Even worse, since funds can be tracked and tainted, no two coins are equal, and fungibility, a fundamental property required in every currency, is at risk. With these threats in mind, several privacy-enhancing technologies have been proposed to improve transaction privacy in Bitcoin. However, they either require a deep redesign of the currency, breaking many currently deployed features, or they address only specific privacy issues and consequently provide only very limited guarantees when deployed separately.
The goal of this work is to overcome this trade-off. Building on CoinJoin, we design ValueShuffle, the first coin mixing protocol compatible with Confidential Transactions, a proposed enhancement to the Bitcoin protocol to hide payment values in the blockchain. ValueShuffle ensures the anonymity of mixing participants as well as the confidentiality of their payment values even against other possibly malicious mixing participants. By combining CoinJoin with Confidential Transactions and additionally Stealth Addresses, ValueShuffle provides *comprehensive privacy* (payer anonymity, payee anonymity, and payment value privacy) without breaking with fundamental design principles or features of the current Bitcoin system. Assuming that Confidential Transactions will be integrated in the Bitcoin protocol, ValueShuffle makes it possible to mix funds of different value as well as to mix and spend funds in the same transaction, which overcomes the two main limitations of previous coin mixing protocols.

1 Introduction

In Bitcoin's initial design, privacy plays only a minor role. The initial perception of Bitcoin providing some built-in anonymity has been refuted by a vast set of academic works [2, 3, 19, 24, 25, 32, 37] showing many different privacy weaknesses with the current Bitcoin protocol. This state of affairs has led to a plethora of privacy-enhancing technologies [3–5,7,16,17,26,34,35,39,42] aiming at overcoming these shortcomings without breaking with the fundamental design of Bitcoin.

However, all of these approaches offer only partial solutions, focusing typically on just one aspect of privacy (payer anonymity, payee anonymity or payment value privacy). For instance, Confidential Transactions (CT) [22], a proposed enhancement to the Bitcoin protocol, which is currently evaluated and tested in the Elements Alpha sidechain [12] and could be implemented in Bitcoin as a soft-fork, defines a transaction format that ensures payment value privacy in the blockchain. Stealth Addresses (SA) [38] is a mechanism for payers to generate unique one-time addresses for improved payee anonymity.

For payer anonymity, the most prevalent approach retaining compatibility with Bitcoin is coin mixing. In a coin mixing protocol, a group of users exchange their coins with each other, effectively hiding the relations between funds and owners. Such functionality can be achieved in practice for example by jointly generating a multi-input multi-output CoinJoin [21] transaction, which enables the users to atomically transfer their funds from potentially tainted inputs to fresh untainted output addresses. Since such a transaction must be signed by each involved user to be valid, theft of funds can easily be avoided. Additionally, if users exchange their output accounts by means of an anonymous broadcast protocol [10, 33, 34], inputs cannot be linked to outputs even by malicious users in the mixing, and such malicious users cannot prevent the honest users from successfully completing the protocol.

To achieve comprehensive privacy, it is necessary to combine all the three aforementioned partial privacy solutions (CT, SA, and mixing) into one solution, but this poses a challenge. SA or other means to generate one-time addresses can be easily combined with coin mixing, but while CT has in fact been designed with CoinJoin mixing in mind, it is not clear that the trust models of CT and P2P coin mixing can be made compatible. The design of CT assumes that a transaction is created by just one user, whereas in P2P coin mixing it is a group of mutually distrusting users who jointly must create a CoinJoin transaction. This leads to the following question:

> Can we design a P2P coin mixing protocol that enables a group of mutually distrusting users to create a CoinJoin confidential transaction, without revealing the relation between inputs and outputs or their payment values to each other?

1.1 ValueShuffle: Mixing Confidential Transactions

In this work, we answer this question affirmatively. We design ValueShuffle, the first coin mixing protocol compatible with CT. ValueShuffle is an extension of the P2P coin mixing protocol CoinShuffle++ [34], which is the result of instantiating the efficient message mixing protocol DiceMix [34] in the setting of CoinJoin-based coin mixing. Since ValueShuffle successfully combines coin mixing, SA and the CT proposal, the resulting currency provides comprehensive privacy, i.e., payer anonymity, payee anonymity and value privacy. Since it builds upon CoinJoin, ValueShuffle inherits a variety of features crucial to its practical deployment in the Bitcoin ecosystem, e.g., compatibility with Bitcoin scripts and compatibility with blockchain pruning.

Exploiting Synergies. By combining coin mixing with SA and CT, we exploit important synergies which make P2P coin mixing both more efficient and more practical, thereby releasing the full potential of coin mixing. We achieve that goal by overcoming the two main limitations of current coin mixing approaches.

First, all forms of coin mixing have been heavily restricted to mixing funds of the same value, because otherwise it is trivial for an observer to link inputs and outputs together just based on their monetary value, independently of how the mixing is organized. Adding value privacy to coin mixing removes this restriction entirely but comes with the challenge of proving to the network that no money is created in the mixing, since payment values are no longer in clear.

Second, current P2P coin mixing protocols [34] suffer from the problem that users are required to mix their funds (in a CoinJoin transaction) by sending them to a fresh address of their own first, which removes the trace to the owner. Only afterwards can users spend the mixed funds to a payee in a second transaction.[1]

This two-step process renders mixing expensive for users, who pay additional fees and need to wait longer, and for the entire Bitcoin network, which has to process essentially twice the amount of transaction data. As a result, privacy comes at a large expense. This is highly undesirable and creates a conflict between privacy and efficiency.

In ValueShuffle, instead, we rely on SA and CT to enable users to send their funds directly to the expected receivers in the CoinJoin transaction, which is arguably the most desirable mode of use of CoinJoin.

1.2 Features of ValueShuffle

The combination of the three privacy-enhancing technologies ValueShuffle, SA, and CT achieves the following main features.

Comprehensive Privacy. The combination of technologies provides the privacy guarantees of interest in Bitcoin. In particular, ValueShuffle ensures that no attacker observing the blockchain or the network, or even participating in the protocol, can link inputs and outputs of the CoinJoin transaction created in an execution of ValueShuffle. That implies that given an output of this transaction, the payer's input address cannot be identified among the honest input addresses in the mixing (payer anonymity). Additionally, SA provides one-time addresses for receiving payments, preventing linkage to the intended payee (payee anonymity), and CT provides value privacy.

Single Transaction. ValueShuffle can be used to transfer funds to payees directly without any form of premixing as required by current P2P coin mixing solutions, and without requiring interaction with the payee. As a result, private payments can be performed with just one single transaction on the blockchain.

[1] This is due to a fundamental restriction [34] of P2P mixing protocols; they can only handle freshly generated messages, which can be discarded if the protocol is disrupted, e.g., Bitcoin addresses of their own generated in the beginning of the protocol. As a result, paying to a payee directly is not possible, because that would require using a fixed amount or a fixed address as a message.

DoS Resistance. ValueShuffle succeeds in the presence of denial-of-service attacks by disruptive users aiming to prevent honest users from completing the mixing. While disruptive users can delay the protocol, they cannot stop it. Since ValueShuffle is based on the efficient CoinShuffle++ protocol [34], it terminates in only $4 + 2f$ communication rounds in the presence of f disruptive users.

Anonymous Channel Not Strictly Required. For providing unlinkability of inputs and outputs in a CoinJoin transaction, ValueShuffle does not rely on any external anonymous channel such as the Tor network [11]. (However, to avoid an observer being able to link inputs of the CoinJoin transaction with network-level identifiers such as IP addresses, using an external means of anonymous communication is highly recommended.)

Features Inherited from CoinJoin. Since ValueShuffle is based on the CoinJoin paradigm, it additionally inherits all of its practical advantages.

Theft Resistance. Since honest users will check the final CoinJoin transaction before signing it, no money can be stolen from them.

Script Compatibility. While ValueShuffle does not keep the scripts confidential, it is compatible with transaction outputs that use complex scripts, e.g., advanced smart contracts, and provides meaningful privacy guarantees for them.

No Overhead for the Network. Unlike ring signatures, as for example deployed in Monero [28], which require a signature of size proportional to the anonymity set, our approach—while requiring interaction between users—provides anonymity without putting an additional burden in terms of blockchain space or verification time on the Bitcoin network.

Reduced Fees and Space Requirements. Taking this one step further, CoinJoin makes Bitcoin in fact more efficient, assuming the availability of Schnorr signatures [36], which are planned to be deployed in Bitcoin Core in the future [6]. The introduction of Schnorr signatures will enable aggregate signatures using an interactive two-round protocol among the users in a CoinJoin [41], reducing the number of signatures from n to 1, where n is the number of users. This protocol can easily be integrated in ValueShuffle, and since we can exploit concurrency, the resulting protocol will have the same number of rounds as the non-interactive variant $(4f + 2)$. This enhancement greatly reduces the size of transactions, thereby providing large savings in terms of blockchain space, verification time, and transaction fees as compared to n individual confidential transactions.

Incentive for Privacy. Due to the reduced fees, users save money by performing privacy-preserving transactions. This provides an unprecedented incentive for deployment and use of privacy-enhancing technologies in Bitcoin.

Compatibility with Pruning. Unlike in Zerocash [4] or Monero [28], using CoinJoin it can be publicly observed which transaction outputs are unspent. While this releases some information to the public, it allows pruning spent outputs from the set of (potentially) unspent transaction outputs. Pruning helps to mitigate the scaling issues of Bitcoin.

Overlay Design. The unlinkability provided by ValueShuffle through the use of CoinJoin is built as a separate layer on top of Bitcoin, which avoids additional complexity and risk in the underlying Bitcoin protocol.

2 Related Work

A variety of privacy solutions have been proposed so far in the literature, based on different paradigms.

Coin Mixing. CoinShuffle [33] and its successor CoinShuffle++ [34] use P2P mixing to create a CoinJoin [21] transaction. This approach has the advantage that theft is excluded by the design; however, efforts are required to ensure termination even with malicious users. ValueShuffle is an extension of CoinShuffle++; both easily scale up to anonymity sets of moderate size (e.g., 50 users).

Another line of research defines mixing using an intermediary tumbler [7, 16, 17, 39]. Notably, TumbleBit is the first such protocol that does not require users to trust in the tumbler for privacy or security of funds. An immediate advantage of mixing with a tumbler is that it scales better to larger anonymity sets. For instance, TumbleBit has been tested with an anonymity set of 800 users. However, a normal payment using TumbleBit needs at least two sequential Bitcoin transactions. ValueShuffle instead needs only one transaction to perform a payment. To enable more efficient mixing, TumbleBit also supports mixing based on payment channels with the tumbler.

Xim [5] uses announcements on the blockchain to pool users for mixing, thereby avoiding that a single party such as the bulletin board in P2P mixing or the tumbler can deny service to honest users and simplify Sybil attacks reducing the effective anonymity set of other honest users. However, Xim supports only two-party mixing, and thus many mixing transactions are required to achieve even an anonymity set of moderate size.

Apart from their differences in terms of requirements (e.g., number of transactions or trust assumptions), all coin mixing protocols proposed thus far are not compatible with CT and thus share the common drawback inherent to coin mixing with plain amounts: payments must transfer the same amount of funds, as otherwise unlinkability of input and output accounts is trivially broken.

CryptoNote. The CryptoNote [35] design is the closest to our work in terms of provided privacy guarantees. CryptoNote relies on ring signatures to provide anonymity for the sender of a transaction. An extension of CryptoNote is fully compatible with CT [27] and has been implemented in the cryptocurrency Monero [28]. In contrast to ValueShuffle, an online mixing protocol is not required, and a sufficient anonymity set can be created using funds of users currently offline.

However, CryptoNote's use of ring signatures comes with two important drawbacks for scalability. First, CryptoNote essentially performs mixing on the blockchain and requires each transaction to contain a ring signature of size $O(n)$, where n is the size of the anonymity set. Storing the ring signatures requires a lot of precious space in the blockchain, and verifying them puts a large burden on all nodes in the currency network. In contrast, ValueShuffle performs the actual mixing off-chain and stores only the result on the blockchain.

Second, CryptoNote is not compatible with pruning, a feature supported, e.g., by the Bitcoin Core client [29]. Pruning reduces the storage requirements of nodes drastically by deleting old blocks and spent transactions once verified. This is impossible in CryptoNote because its use of ring signatures prevents clients from determining whether an transaction output has been spent and can be pruned. A CoinJoin-based approach such as ValueShuffle does not have this problem and is compatible with pruning.

From a high-level point of view, ValueShuffle moves the overhead of providing payer anonymity from the blockchain and thus the whole Bitcoin network to only the users actively involved in a single mixing.

Mimblewimble. Mimblewimble [18, 31] is a design for a cryptocurrency with confidential transactions that can be aggregated non-interactively and even across blocks. This has tremendous benefits for the scalability of the underlying blockchain. However, such aggregation alone does not ensure input-output unlinkability against parties who perform the aggregation, e.g., the miners. Furthermore, Mimblewimble is not compatible with smart contracts due to the lack of script support. In contrast, ValueShuffle seamlessly supports scripts as currently implemented in Bitcoin.

Zerocoin and Zerocash. Zerocoin [26] and its follow-up work Zerocash [4], whose implementation Zcash has recently been deployed, are cryptocurrency protocols that provide anonymity by design. Although these solutions provide strong privacy guarantees, it is not clear whether Zcash will see widespread adoption, in particular given its reliance on a trusted setup and non-falsifiable cryptographic assumptions [14] due to the use of zkSNARKS. Moreover, since it is not possible to observe which outputs have been spent already, blockchain pruning is not possible in Zerocoin and Zerocash.

3 Building Blocks

We describe the three building blocks of ValueShuffle, namely peer-to-peer mixing, Confidential Transactions, and Stealth Addresses.

3.1 Peer-to-Peer Mixing

A peer-to-peer (P2P) mixing protocol [10, 33, 34] allows a set of untrusted users to simultaneously broadcast their messages without requiring any trusted third party. The protocol ensures *sender anonymity*, i.e., an attacker controlling the network and some of the participating users cannot associate a message to its corresponding honest sender. In this work, we use DiceMix [34] (as in CoinShuffle++), which relies on Dining Cryptographers networks (DC-nets) [9] to achieve anonymity. Before the DC-net can be run, DiceMix runs a key exchangeto establish pairwise symmetric keys.

DoS Resistance. Disruptive users, whose goal is to prevent honest users from mixing, will be exposed and excluded in DiceMix. To this end, users broadcast the ephemeral secret key used in the key exchange of a failed protocol run. Then at least one malicious user is identified by replaying the expected computations, allowing the honest users to start a new run excluding the malicious user. Eventually only honest users remain, and the protocol terminates.

Freshness of Messages. A P2P mixing protocol requires mixed messages to be fresh and to have sufficient entropy [34]. Freshness enables the protocol to sacrifice anonymity in failed runs in order to identify malicious users. As the messages will be discarded and no transaction will be performed, this does not hurt privacy at this point. Freshness is then required to ensure that a message from a particular user used in a failed run (with sacrificed anonymity) cannot be linked to a message from the same user used in the final successful run, in which a transaction will be performed and anonymity must be ensured.

3.2 Confidential Transactions

Confidential Transactions (CT) [15, 22] is a cryptographic extension to Bitcoin that allows a single user to perform a transaction such that none of the monetary values in the inputs or outputs are revealed, thereby guaranteeing *value privacy*. Nevertheless, the balance property, i.e., no new coins are generated in the transaction, remains publicly verifiable.

This is mainly achieved by hiding the values using additively homomorphic commitments, i.e., $\mathsf{Com}(x,r) \oplus \mathsf{Com}(x',r') = \mathsf{Com}(x+x',r+r')$. As an example, assume a user has an input value x_1 and two output values x_2 and x_3. She can commit to x_1, x_2, and x_3, as $c_i := \mathsf{Com}(x_i, r_i)$, where r_i is chosen uniformly at random. Then, she computes $r_\Delta = r_1 + r_2 - r_3$ and adds this value in clear to the transaction. Ignoring fees, a verifier can then verify the balance property by checking whether $c_1 \oplus c_2 = c_3 \oplus \mathsf{Com}(0, -r_\Delta)$.

In fact, the current design of CT avoids adding r_Δ explicitly by choosing the randomness values such that always $r_\Delta = 0$. Our description of ValueShuffle is not compatible with this optimization, because it is not clear how to support it without adding communication rounds to the protocol. In practice, CT uses Pedersen commitments [30] and range proofs based on borromean ring signatures [23].

To ensure that commitments do not contain negative or too-large values that could overflow, a non-interactive zero-knowledge *range proof* is added to every commitment, proving that the value is in a certain range. (Also other components, e.g., an ephemeral public key, are added. To simplify presentation, we assume throughout the paper that these other components are part of the range proof.)

Monetary values are in fact represented not as integers but as floating point numbers with a public exponent, and only the mantissa is hidden in the commitment; this is to support large values efficiently. However, in this work we assume ordinary integers, i.e., we assume that the exponent is effectively not used (i.e., it is always 0), which will be necessary to ensure anonymity.

3.3 One-time Addresses

Users performing transactions via ValueShuffle require a sufficient supply of fresh unlinkable addresses of the payee. This will make it possible to discard a recipient address used in a failed run of DiceMix. In this case, a fresh address can be used for the following run, satisfying the freshness requirement of messages mixed using DiceMix. (If there are n users in the mixing, DiceMix will require at most $n - 1$ addresses.) Several methods are available. First, the payee can post a stealth address, which enables any payer to derive fresh addresses. Second, the payee can send a BIP32 public key [40] to the payer, which enables the payer to derive fresh addresses. The necessary derivation index can be derived from public information, e.g., a hash of the value commitment. Third, the payee can simply send enough fresh addresses to the payer.

The method based on stealth addresses provides the strongest privacy guarantees. A stealth address is a public long-term address of a payee, which enables a payer to derive an arbitrary number of unlinkable addresses owned by the payee. A payment using a stealth address does not require any direct communication between payer and payee, and thus provides strong payer anonymity when used together with coin mixing: not even the payee can identify the payer, which is a useful property for anonymous donations, for example.

Nevertheless, ValueShuffle is oblivious of the method to generate fresh addresses; we only require that the payee has access to some method, and we refer the reader to the respective descriptions of the individual methods for details.

4 ValueShuffle

In this section, we overview ValueShuffle, the first P2P coin mixing protocol compatible with CT. We detail the protocol and the security analysis in Appendix A.

Bootstrapping and Communication Model. A suitable bootstrapping mechanism is required for finding users. A malicious bootstrapping mechanism can hinder payer anonymity, as it can prevent honest users from participating in the protocol. Although this is a realistic threat, we consider prevention of such attacks orthogonal to our work.

Since ValueShuffle is an extension of CoinShuffle++, which uses DiceMix to mix Bitcoin addresses of the users, we rely on the same communication model as CoinShuffle++ and DiceMix. For completeness, we sketch this model here. We assume that users communicate with each other via a (broadcast) bulletin board, e.g., an IRC server echoing messages from one user to the others. Moreover, we assume the bounded synchronous communication setting, where a message from a user is available to all others at the end of a round and absence of a message from a user indicates that the user did not send any message. We stress that privacy is guaranteed even against a fully malicious bulletin board; the bulletin board is purely a means of communication.

4.1 Security and Privacy Features

ValueShuffle provides the following security and privacy guarantees.

Unlinkability: Given an output and two inputs belonging to honest users in the CoinJoin transaction created by the protocol, the attacker is not able to tell which of the two inputs pays to the output.

CT Compatibility: The protocol generates a CoinJoin transaction without compromising the individual value privacy of honest users provided by CT.

Theft Resistance: Funds of each honest user are either transferred to the payee as intended or remain with the honest user.

Termination: The protocol terminates for the honest users.

Threat Model. We consider an attacker that controls f malicious users. We do not put restrictions on f. However, for unlinkability we need $f < n - 1$, where n is the set of unexcluded users, to ensure there is a meaningful anonymity set.

The attacker additionally controls the bulletin board, which enables him to block messages from honest users. Only for the termination property, we assume that the bulletin board is honest, because otherwise, all communication could be blocked by the attacker and termination is impossible to ensure.

4.2 Challenges and Our Solutions

To combine coin mixing with CT and one-time addresses, we need to overcome the following challenges. For the sake of explanation, we assume that each user has only one input and one output in the transaction, and that there is no transaction fee. The full protocol does not have these limitations.

Basic Design. From a high-level point of view, the users in an execution of ValueShuffle run DiceMix to mix not only their output addresses (as done in CoinShuffle++) but their *output triples*, i.e., triples consisting of output address (or script), CT value commitment, and corresponding range proof. If DiceMix runs successfully, then it will pass a set of anonymized triples to an application-defined *confirmation* mechanism, which confirms the result of the mixing. As in CoinShuffle++, the confirmation mechanism in ValueShuffle is the collective signing of the CoinJoin transaction, either by collecting a plain list of signatures or by performing an interactive protocol to create an aggregate Schnorr signature [41].

Handling Disruption. If a run of DiceMix fails, it must be possible to identify at least one disruptive user to be excluded in a subsequent run of the protocol. This will eventually guarantee termination. Crucially, DiceMix requires the confirmation mechanism to output at least one such user if confirmation itself is disrupted. The confirmation mechanism can assume that the result of the mixing is correct, i.e., it contains the messages of all honest users. Given that assumption, identifying a disruptive user is straightforward: a user that refuses to sign the final CoinJoin transaction, or provides wrong signatures (or wrong partial signatures in the case of Schnorr aggregate signatures) is obviously disruptive.

Freshness of Mixed Output Triples. Recall that DiceMix requires mixed messages (i.e., the output triples in our case) to be fresh [34] and have sufficient entropy to ensure anonymity. This is exactly where we are able to exploit one-time addresses and CT. In particular, the payer is able to create fresh unlinkable output triples: We assume that the payer has a method to create fresh unlinkable output addresses all belonging to same payee, so the address component of the output triple is fresh. Moreover, the payer uses CT and since the commitment scheme and the range proof are randomized, the payer is able to generate many fresh unlinkable value commitments and range proofs. So all three components of the output triple can be freshly generated.

Only by combining one-time addresses and CT, we are able to guarantee anonymity if users are mixing and performing actual payments in the same transaction. Previous P2P coin mixing protocols such as CoinShuffle++ [34] require users to mix funds to a fresh output address of their own, because using the fixed address of the recipient or even using the plain monetary value in the mixing is not possible if anonymity and termination are desired [34].

Multiple-payer CT. In the original design of CT, the single payer can easily craft the randomness for the commitments to input and output values in the transaction such that anyone can verify its correctness (see Section 3.2). However, in a mixing transaction with several payers, a naive construction of such a verifiable transaction would require that users reveal to each other the randomnesses used in the commitments, thereby compromising the hiding property of the commitments.

To overcome this issue, the users can run a secure sum protocol to jointly compute the sum r_Δ of their random values, i.e., $r_\Delta = \sum_i r_i - r'_i$, where r_i denotes the randomness in the commitment to the input value of user i, and r'_i denotes the randomness in the commitment to the output value of the same user i. As a sum, r_Δ does not reveal which user contributed which summand to r_Δ. Now all users can add r_Δ as an explicit public randomness value to the transaction, and the overall transaction is valid again, which can be publicly verified by checking whether $\bigoplus c_i = \bigoplus c'_i \oplus \mathsf{Com}(0, -r_\Delta)$.

The value r_Δ can be obtained with a standard secure sum protocol based on additive secret sharing, where every user i broadcasts her value $r_i - r'_i$ blinded by multiple pads, each one shared with one other user. The messages from all users are then combined so that shared keys cancel out and the sum r_Δ is obtained. This mechanism is in essence equivalent to a DC-net as already used in DiceMix.

Handling Disruption of the Secure Sum Protocol. Malicious users can disrupt not only the mixing of output triples but also the secure sum protocol by creating an output value commitment that does not match the value of the input commitment, which can be detected when creating the CoinJoin transaction.

Similar to sacrificing anonymity in the DC-net used for mixing output triples, we can sacrifice anonymity in the DC-net used as a secure sum protocol. This reveals for every user i what she claims is $r_i - r'_i$. Using the verification equation of CT, all honest users can easily check the balance property of every user i individually, i.e., check whether user i's output commitments are consistent with her input commitments. Note that this approach does not reveal the random values used for the input commitments or the output commitments of user i, which would also reveal her intended payment value.

Combining P2P Mixing and Secure Sum. Since both DiceMix and the secure sum protocol are similar in structure (they both rely on DC-nets after all), we can optimize their combination. First, we can rely on a single key exchange and derive independent subkeys for the P2P mixing and the secure sum protocol. This means that if one of the two protocols is aborted, then the other must be aborted as well, because the same ephemeral secret is used for the key exchange and must be revealed. This is not a problem because the proper result of one of the two protocols does not yield a valid mixing transaction, and the users have to restart from scratch by generating a fresh output triple anyway.

Mixing Long Messages in DiceMix. While DiceMix in its current form is practical for small messages m (e.g., $|m| = 160$ bits as used by CoinShuffle++), it is prohibitively slow for messages of the size we require; we need $|m| \approx 20\,000$ bits to mix the quite large range proofs necessary in CT. The most expensive computation step is a polynomial factorization and requires each message to be an element of a finite field and consequently the finite field must have a size of about $2^{|m|}$.

To overcome this issue, we split m into several chunks, i.e., $m = m_1 \parallel \cdots \parallel m_\ell$ and mix those chunks in different parallel runs of the essential mixing step in DiceMix. The challenge that arises is to recombine the messages again, because the mixing ensures that it is not possible to know which chunks belong together (i.e., to the same user). Our solution is to prefix every m_i for $1 < i \leq \ell$ with $\mathsf{F}(m_1)$, where F is a collision-resistant hash function, so that every user mixes: $m'_1 = m_1$ and $m'_i = \mathsf{F}(m_1) \parallel m_i$ for $1 < i \leq \ell$. This arrangement allows for a trade-off between computation and communication required for mixing: bigger chunks reduce the number of parallel mixing instances required but demand higher computation costs for the polynomial factorization.

Supporting Arbitrary Scripts. So far we have discussed only output addresses, which are essentially hashes, but not about their type. While mixing works fine with ordinary pay-to-pubkey-hash (P2PKH) hashes, we require pay-to-script-hash (P2SH) hashes [1][2] to support arbitrary scripts. However, it is not possible to mix P2PKH and P2SH hashes in the same mixing, because this would require adding the address type explicitly to the mixing message, which breaks anonymity: in case of a disruption, it becomes clear which inputs go a P2PKH address and which inputs go to a P2SH address. To support P2PKH and P2SH together, we can instead perform P2PKH transactions nested in P2SH.[3] For simplicity, we will ignore this issue and assume addresses in the remainder of the paper.

[2] In P2PKH, funds are sent to a public key specified by its hash, and the user who wants to spend the resulting output is responsible for showing the public key. P2SH is a generalization: In P2SH, funds are sent to a script specified by its hash, and the user who wants to spend the resulting output is responsible for providing the script.

[3] Such nesting has also been proposed in the context of Segregated Witness [20].

4.3 Overview of ValueShuffle

We assume that every user i is represented by a triple $in_i = (c_i = \mathsf{Com}(x_i, r_i), \pi_i, vk_i)$, where c_i denotes the commitment to the input value x_i using randomness r_i, π_i denotes a range proof for c_i, and vk_i denotes a Bitcoin address owned by the user i. For ease of explanation, we assume here that every user has only one input triple and that there are no fees in place.

From a high-level perspective, an execution of ValueShuffle consists of *runs*, and each run of ValueShuffle consists of four phases as follows.

1. Output Generation. Every user i locally generates her output triple $out_i = (c'_i = \mathsf{Com}(x'_i, r'_i), \pi'_i, addr'_i)$, where c'_i is a CT-style commitment, π'_i is the corresponding range proof, and $addr'_i$ is a fresh one-time address of the receiver. Note that users can have several output triples (including change outputs), but for simplicity we restrict our attention to only one output here.

2. Mixing and Secure Sum. Users run in parallel a P2P mixing protocol to mix their output triples out_i and a secure sum protocol to privately compute the sum $r_\Delta = \sum_i r'_i - r_i$. Finally, input and output messages can be combined to deterministically form a (still unsigned) CoinJoin transaction by adding the explicit random value r_Δ.

3. Check. Users check validity of the resulting CoinJoin transaction, i.e., they check whether all range proofs π'_i verify with respect to commitments c'_i, and check whether the overall balance of the intended transaction is correct, i.e., whether $\bigoplus_i c_i = \bigoplus_i c'_i + \mathsf{Com}(0, -r_\Delta)$. Also, every user verifies that her output triple is part of the mixing result, i.e., no coins are stolen by the transaction.

4a. Confirm. If all checks pass, the transaction is valid and users are required to sign it. While every user checked only that her output is present, DiceMix guarantees that this suffices to ensure that the outputs of all users are present. Thus if some honest user reached this point, she can be sure that users refusing to sign the transaction are disruptive. If this happens, they will be excluded and a new run of the protocol is started.

4b. Blame. If any of the aforementioned checks fail, a blame phase is performed to detect at least one malicious user. Every user i broadcasts the secrets she used for the mixing and secure sum protocols, thereby revealing the value $r_i - r'_i$, which suffices to check that user i committed the same value in the input and output addresses (and therefore no coins were created). Now every other user j can recompute the mixing and secure sum steps of user i and detect whether she faithfully followed the protocol specification. The thereby exposed malicious user is then excluded from the protocol and a new run is started.

4.4 Performance

To reduce the number of necessary communication rounds, DiceMix is able to start a subsequent run even if the current run has not yet failed, and thus even if it is not yet clear who will be the disruptor to be excluded in the subsequent run [34]. ValueShuffle is able to exploit this mechanism as well and as a result, ValueShuffle terminates in $4 + 2f$ rounds in the presence of f disrupting users (instead of $4 + 4f$ rounds without this feature).

Acknowledgements. We thank Pieter Wiulle for insightful discussions and in particular for pointing out a mistake in a preliminary version of this work, and we thank the anonymous reviewers for their very helpful comments. This work was supported by the German Ministry for Education and Research (BMBF) through funding for the German Universities Excellence Initiative.

A ValueShuffle: Full Protocol Description

In this section we specify ValueShuffle fully. We start by describing the building blocks that the protocol relies on.

Digital Signatures. We require a digital signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ unforgeable under chosen-message attacks ($\mathsf{UF\text{-}CMA}$). The algorithm KeyGen returns a private signing key sk and the corresponding public verification key vk. On input message m, $\mathsf{Sign}(sk, m)$ returns σ, a signature on message m using signing key sk. The verification algorithm $\mathsf{Verify}(pk, \sigma, m)$ outputs *true* iff σ is a valid signature for m under the verification key vk.

Non-Interactive Key Exchange. We require a non-interactive key exchange (NIKE) mechanism $(\mathsf{NIKE.KeyGen}, \mathsf{NIKE.SharedKey})$ secure in the CKS model [8,13]. The algorithm $\mathsf{NIKE.KeyGen}(id)$ outputs a public key npk and a secret key nsk for a given party identifier id. $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid)$ outputs a shared key for the two parties and session identifier sid. $\mathsf{NIKE.SharedKey}$ must fulfill the standard correctness requirement that for all session identifiers sid, all parties id_1, id_2, and all corresponding key pairs (npk_1, nsk_1) and (npk_2, nsk_2), it holds that $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid) = \mathsf{NIKE.SharedKey}(id_2, id_1, nsk_2, npk_1, sid)$. Additionally, we require an algorithm $\mathsf{NIKE.ValidatePK}(npk)$, which outputs *true* iff npk is a public key in the output space of $\mathsf{NIKE.KeyGen}$, and we require an algorithm $\mathsf{NIKE.ValidateKeys}(npk, nsk)$ which outputs *true* iff nsk is a secret key for the public key npk.

Static Diffie-Hellman key exchange satifies these requirements [8], given a suitable key derivation algorithm such as $\mathsf{NIKE.SharedKey}(id_1, id_2, x, g^y) := \mathsf{K}((g^{xy}, \{id_1, id_2\}, sid))$ for a hash function K modeled as a random oracle.

Hash Functions. We require three hash functions H, G, and F modeled as random oracles.

Confidential Transactions. Confidential Transactions (CT) relies on a non-interactive commitment scheme $(\mathsf{Com}, \mathsf{Open})$, which uses public parameters we keep implicit, and a range proof $(\mathsf{RPCreate}, \mathsf{RPVerify})$. Algorithm $c := \mathsf{Com}(m, r)$ uses the randomness $r \in \mathcal{R}$ to output a commitment c of message m. Algorithm $b := \mathsf{Open}(param, c, m, r)$ returns *true* iff c is a valid commitment of message m with randomness r. Informally, a commitment scheme is *hiding*, i.e., the commitment c reveals nothing about m; and *binding*, i.e., no attacker can produce a commitment that it can open to two different messages $m' \neq m$. CT requires an additively homomorphic commitment scheme. A commitment scheme is additively homomorphic if there is an efficient operation $\oplus$ on commitments such that $\mathsf{Com}(m_1, r_1) \oplus \mathsf{Com}(m_2, r_2) = \mathsf{Com}(m_1 + m_2, r_1 + r_2)$. In practice, CT uses Pedersen commitments [30] as the instantiation of such a commitment scheme.

In a range proof scheme, the algorithm $\pi := \mathsf{RPCreate}(m, r)$ creates a proof π that $c = \mathsf{Com}(m, r)$ is a commitment of a value in a valid range. The algorithm $b := \mathsf{RPVerify}(\pi, c)$ returns *true* iff π is a valid range proof for c. We refer the reader to the CT draft [15, 22] for details.

Confirmation. The confirmation subprotocol $\textsc{ConfirmTx}()$ uses CoinJoin to perform the actual mixing. The algorithm $\mathsf{CoinJoinTx}()$ creates a CoinJoin transaction, and the algorithm $\mathsf{Submit}(tx, \sigma[\,])$ submits transaction tx including the corresponding signatures $\sigma[\,]$ to the Bitcoin network.

Our implementation of $\textsc{ConfirmTx}()$ produces a CoinJoin transaction with one signature from each user. As noted above, alternative schemes are possible, e.g., the two-round aggregate Schnorr signature protocol [41] can be used to sign the transaction if Schnorr signatures will available in Bitcoin. In that case, it is possible to pre-perform the first round of the two-round protocol (in parallel to the main part of ValueShuffle), because this round does not depend on the output of the mixing, such that $\textsc{ConfirmTx}()$ effectively still takes only one round, and the full protocol still takes only $4 + 2f$ rounds.

Conventions and Notation for Pseudocode. We use arrays written as $\textsc{Arr}[i]$, where i is the index. We denote the full array (all its elements) as $\textsc{Arr}[\,]$.

Message x is broadcast using "**broadcast** x". The command "**receive** $\textsc{x}[p]$ **from all** $p \in P$ **where** $X(\textsc{x}[p])$ **missing** $C(P_{\mathit{off}})$" attempts to receive a message from all users $p \in P$. The first message $\textsc{x}[p]$ from user p that fulfills predicate $X(\textsc{x}[p])$ is accepted and stored as $\textsc{x}[p]$; all further messages from p are ignored. When a timeout is reached, the command C is executed, which has access to a set $P_{\mathit{off}} \subseteq P$ of users that did not send a (valid) message.

Regarding concurrency, a thread that runs a procedure $\mathrm{P}(\mathit{args})$ is started using "$t :=$ **fork** $\mathrm{P}(\mathit{args})$", where t is a handle for the thread. A thread t can either be joined using "$r :=$ **join** t", where r is its return value, or it can be aborted using "**abort** t". A thread can wait for a notification and receive a value from another thread using "**wait**". The notifying thread uses "**notify** t **of** v" to notify thread t of some value v.

Setup. We assume that funds that should be used as input in ValueShuffle can only be spent by providing signatures, i.e., they are associated with a verification key that can also be used in ValueShuffle. Furthermore, for ease of explanation we assume here that every user has only one input. However, ValueShuffle can easily be adapted to overcome this restriction: If a user has more than one input, she can simply sign her messages using all signing keys corresponding to all verification keys, and the code for checking the balance can be adapted to consider the homomorphic combination of several input commitments.

As a result of these assumptions, every user in the beginning knows an unspent transaction output UTXO[p], its corresponding CT commitment C[p] and verification key VK[p] for every other user p.

Furthermore, every user has her corresponding secrets, i.e., the value x and randomness r such that $c = \mathsf{Com}(x, r)$, the secret key sk corresponding to vk, and every user has a set *Payments* with recipients and corresponding amounts (including a change address if necessary), describing the payments she wants to perform. We assume that every user wants to perform the same number of payments and that the transaction fee *fee* is evenly split among the users.

Full Pseudocode. Here we describe the full protocol in pseudocode. We assume that the reader is familiar with the details of DiceMix and CoinShuffle++ [34] to understand the code. For better readability, our essential changes to CoinShuffle++, which result in ValueShuffle, are printed in blue.

```
1: procedure ValueShuffle($P, my, \mathrm{VK}[\,], sk, \mathrm{UTXO}[\,], \mathrm{C}[\,], r, Payments, sid$)
2:     $sid := (sid, P, \mathrm{VK}[\,])$
3:     if $my \in P$ then
4:         fail “cannot run protocol with myself”
5:     return Run($P, my, \mathrm{VK}[\,], sk, sid, 0$)
6: procedure Run($P, my, \mathrm{VK}[\,], sk, sid, run$)
7:     if $P = \emptyset$ then
8:         fail “no honest users”
9:     ▷ Exchange pairwise keys
10:    $(\mathrm{NPK}[my], \mathrm{NSK}[my]) := \mathsf{NIKE.KeyGen}(my)$
11:    $sidPre := \mathsf{H}((\mathtt{sidPre}, sid, run))$
12:    broadcast $(\mathtt{KE}, \mathrm{NPK}[my], \mathsf{Sign}(sk, (\mathrm{NPK}[my], sidPre)))$
13:    receive $(\mathtt{KE}, \mathrm{NPK}[p], \sigma[p])$ from all $p \in P$
14:        where $\mathsf{NIKE.ValidatePK}(\mathrm{NPK}[p])$
                 $\wedge\ \mathsf{Verify}(\mathrm{VK}[p], \sigma[p], (\mathrm{NPK}[p], sidPre))$
15:    missing $P_{off}$ do
16:        $P := P \setminus P_{off}$                         ▷ Exclude offline users
17:    $sid' := \mathsf{H}((\mathtt{sid'}, sid, P \cup \{my\}, \mathrm{NPK}[\,], run))$
18:    $\mathrm{K}[\,] := \textsc{DC-Keys}(P, \mathrm{NPK}[\,], my, \mathrm{NSK}[my], sid')$
19:    ▷ Generate fresh outputs to mix
20:    $(myOut, myr) := \textsc{GenOutputs}(Payments)$
21:    $\mathrm{DC}[my][\,][\,][\,] := \textsc{DC-Mix}(P, my, \mathrm{K}[\,], myOut)$
22:    $\textsc{SumDC}[my] := myr + \textsc{DC-Slot-Pad}(P, my, \mathrm{K}[\,], \mathtt{sum})$
23:    $P_{ex} := \emptyset$           ▷ Malicious (or offline) users for later exclusion
24:    ▷ Commit to DC-net vector
25:    $\textsc{Com}[my] := \mathsf{H}((\mathtt{CM}, \mathrm{DC}[my][\,][\,][\,], \textsc{SumDC}[my]))$
26:    broadcast $(\mathtt{CM}, \textsc{Com}[my], \mathsf{Sign}(sk, (\textsc{Com}[my], sid')))$
27:    receive $(\mathtt{CM}, \textsc{Com}[p], \sigma[p])$ from all $p \in P$
28:        where $\mathsf{Verify}(\mathrm{VK}[p], \sigma[p], (\textsc{Com}[p], sid'))$
29:    missing $P_{off}$ do                    ▷ Store offline users for exclusion
30:        $P_{ex} := P_{ex} \cup P_{off}$
31:    if $run > 0$ then
```

```
32:         ▷ Wait for prev. run to notify us of malicious users
33:         $P_{exPrev} :=$ **wait**
34:         $P_{ex} := P_{ex} \cup P_{exPrev}$

35:     ▷ Collect shared keys with excluded users
36:     **for all** $p \in P_{ex}$ **do**
37:         $\mathrm{K}_{ex}[my][p] := \mathrm{K}[p]$
38:     ▷ Start next run (in case this one fails)
39:     $P := P \setminus P_{ex}$
40:     $next :=$ **fork** $\textsc{Run}(P, my, \mathrm{VK}[\,], sk, sid, run + 1)$

41:     ▷ Open commitments and keys with excluded users
42:     **broadcast** $(\mathtt{DC}, \mathrm{DC}[my][\,][\,][\,], \textsc{SumDC}[my], \mathrm{K}_{ex}[my][\,], \mathsf{Sign}(sk, \mathrm{K}_{ex}[my][\,]))$
43:     **receive** $(\mathtt{DC}, \mathrm{DC}[p][\,][\,][\,], \textsc{SumDC}[p], \mathrm{K}_{ex}[p][\,], \sigma[p])$ **from all** $p \in P$
44:         **where** $\mathsf{H}((\mathtt{CM}, \mathrm{DC}[p][\,][\,][\,], \textsc{SumDC}[p])) = \textsc{Com}[p]$
                    $\wedge \{p' : \mathrm{K}_{ex}[p][p'] \neq \bot\} = P_{ex}$
                    $\wedge \mathsf{Verify}(\mathrm{VK}[p], \mathrm{K}_{ex}[p][\,], \sigma[p])$
45:     **missing** $P_{off}$ **do**                              ▷ Abort and rely on next run
46:         **return** $\textsc{Result-Of-Next-Run}(P_{off}, next)$
47:     $Out := \textsc{DC-Mix-Res}(P \cup \{my\}, \mathrm{DC}[\,][\,][\,][\,], P_{ex}, \mathrm{K}_{ex}[\,][\,])$
48:     $r_{\triangle} := \textsc{DC-Slot-Open}(P \cup \{my\}, \textsc{SumDC}[\,], \mathtt{sum}, P_{ex}, K_{ex}[\,][\,])$

49:     ▷ Check if our output is contained in the result
50:     $(balanced, Out_{mal}) := \textsc{VerifyResult}(i, P, Out, \mathrm{C}[\,], r_{\triangle})$
51:     **if** $myOut \subseteq Out \wedge balanced \wedge Out_{mal} = \emptyset$ **then**
52:         $P_{mal} := \textsc{ConfirmTx}(i, P, Out, my, \mathrm{VK}[\,], sk, \mathrm{UTXO}[\,], sid)$
53:         **if** $P_{mal} = \emptyset$ **then**                                            ▷ Success?
54:             **abort** $next$
55:             **return** $m$
56:     **else**
57:         **broadcast** $(\mathtt{SK}, \mathrm{NSK}[my])$                                ▷ Reveal secret key
58:         **receive** $(\mathtt{SK}, \mathrm{NSK}[p])$ **from all** $p \in P$
59:             **where** $\mathsf{NIKE.ValidateKeys}(\mathrm{NPK}[p], \mathrm{NSK}[p])$
60:         **missing** $P_{off}$ **do**                          ▷ Abort and rely on next run
61:             **return** $\textsc{Result-Of-Next-Run}(P_{off}, next)$
62:         ▷ Determine malicious users using the secret keys
63:         $P_{mal} := \textsc{Blm}(P, \mathrm{NPK}[\,], my, \mathrm{NSK}[\,], \mathrm{DC}[\,][\,][\,][\,], sid', P_{ex}, \mathrm{K}_{ex}[\,][\,], Out_{mal}, \mathrm{C}[\,])$
64:     **return** $\textsc{Result-Of-Next-Run}(P_{mal}, next)$

65: **procedure** $\textsc{DC-Mix}(P, my, \mathrm{K}[\,], myM)$
66:     $o := 1$
67:     **for all** $m \in myM$ **do**
68:         ▷ Split message into chunks and prefix those
69:         $\mathrm{C}[1] \parallel rem := m$            ▷ C[1] must be long enough to be unpredictable
70:         $\mathrm{C}[2] \parallel \ldots \parallel \mathrm{C}[n] := rem$         ▷ $\forall j \in \{2, \ldots, n\}. |\mathrm{C}[j]| = |\mathrm{C}[1]| - |\mathsf{F}(\mathrm{C}[1])|$
71:         **for** $j := 2, \ldots, n$ **do**
72:             $\mathrm{C}[j] := \mathsf{F}(\mathrm{C}[1]) \parallel \mathrm{C}[j]$

73:         ▷ Create power sums in individual slots
74:         **for** $j := 1, \ldots, n$ **do**
```

```
            **for** $i := 1, \ldots, |P| + 1$ **do**[4]
                $\textsc{DCMy}[o][j][i] := \mathrm{C}[j]^i + \textsc{DC-Slot-Pad}(P, \mathit{my}, \mathrm{K}[\,], (o, j, i))$
        $o := o + 1$
    **return** $\textsc{DCMy}[\,][\,][\,]$

**procedure** $\textsc{DC-Mix-Res}(P_{all}, \textsc{DCMix}[\,][\,][\,], P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    $n := |\textsc{DCMix}[1]|$
    **for** $j := 1, \ldots, n$ **do**
        **for** $s := 1, \ldots, |P_{all}|$ **do**
            $\mathrm{M}^*[s] := \textsc{DC-Slot-Open}(P_{all}, \textsc{DCMix}[\,][j][\,], s, P_{ex}, \mathrm{K}_{ex}[\,][\,])$
        ▷ Solve equation system for array M[ ] of messages
        $\mathrm{M}[j][\,] := \mathsf{Solve}(\forall s \in \{1, \ldots, |P_{all}|\}.\ \mathrm{M}^*[s] = \sum_{i=1}^{|P_{all}|} \mathrm{M}[i]^s)$
    ▷ Recombine messages
    $M := \emptyset$
    **for** $i := 1, \ldots, |P_{all}|$ **do**
        $m := \mathrm{M}[1][i]$
        **for** $j := 2, \ldots, n$ **do**
            $S := \{(i, m^*) : h \parallel m^* = \mathrm{M}[j][i] \wedge h = \mathsf{F}(\mathrm{M}[1][i])\}$
            ▷ Unique match?
            **if** $\exists i, m^*.\ S = \{(i, m^*)\}$ **then**
                $m := m \parallel m^*$
            **else**
                **continue** (outer loop)                    ▷ Invalid encoding, ignore message
        $M := M \cup \{m\}$
    **return** $M$

**procedure** $\textsc{DC-Slot-Pad}(P, \mathit{my}, \mathrm{K}[\,], s)$
    **return** $\sum_{p \in P} \mathrm{sgn}(\mathit{my} - p) \cdot \mathsf{G}((\mathrm{K}[p], s))$                    ▷ in $\mathbb{F}$

**procedure** $\textsc{DC-Slot-Open}(P_{all}, \mathrm{DC}[\,][\,], s, P_{ex}, \mathrm{K}_{ex}[\,][\,])$
    ▷ Pads cancel out for honest users
    $m^* := \sum_{p \in P_{all}} \mathrm{DC}[p][s]$                    ▷ in $\mathbb{F}$
    ▷ Remove pads for excluded users
    $m^* := m^* - \sum_{p \in P_{all}} \textsc{DC-Slot-Pad}(P_{ex}, p, \mathrm{K}_{ex}[p][\,], s)$
    **return** $m^*$

**procedure** $\textsc{DC-Keys}(P, \mathrm{NPK}[\,], \mathit{my}, \mathit{nsk}, \mathit{sid}')$
    **for all** $p \in P$ **do**
        $\mathrm{K}[p] := \mathsf{NIKE.SharedKey}(\mathit{my}, p, \mathit{nsk}, \mathrm{NPK}[p], \mathit{sid}')$
    **return** $\mathrm{K}[\,]$

**procedure** $\textsc{Blm}(P, \mathrm{NPK}[\,], \mathit{my}, \mathrm{NSK}[\,], \mathrm{DC}[\,][\,][\,][\,], \mathit{sid}', P_{ex}, \mathrm{K}_{ex}[\,][\,], \mathit{Out}_{mal}, \mathrm{C}[\,])$
    $P_{mal} := \emptyset$
    **for all** $p \in P$ **do**
        $P' := (P \cup \{\mathit{my}\} \cup P_{ex}) \setminus \{p\}$
        $\mathrm{K}'[\,] := \textsc{DC-Keys}(P', \mathrm{NPK}[\,], p, \mathrm{NSK}[p], \mathit{sid}')$
        ▷ Reconstruct purported message $m'$ of $p$
        **for** $o := 1, \ldots, |\mathrm{DC}[\mathit{my}]|$ **do**
```

[4] If $\mathit{run} > 0$, it suffices to loop up to $|P|$, because at least one user will have been excluded when the DC-net is opened.

```
            $m' := \mathrm{DC}[p][o][1][1] - \textsc{DC-Slot-Pad}(P', p, \mathrm{K}'[\,], (o, 1, 1))$
            for $j := 2, \ldots, |(\mathrm{DC}[p][o]|$ do
                $m^* \parallel h := \mathrm{DC}[p][o][j][1] - \textsc{DC-Slot-Pad}(P', p, \mathrm{K}'[\,], (o, j, 1))$
                $m' := m' \parallel m^*$
            $Out' := Out' \cup \{m'\}$
        ▷ Replay DC-net messages of $p$
        $\mathrm{DC}'[\,][\,][\,] := \textsc{DC-Mix}(P', p, \mathrm{K}'[\,], Out')$
        if $\mathrm{DC}'[\,][\,][\,] \neq \mathrm{DC}[p][\,][\,][\,]$ then                ▷ Exclude inconsistent $p$
            $P_{mal} := P_{mal} \cup \{p\}$
        ▷ Verify that $p$ has sent valid range proofs
        if $Out' \cap Out_{mal} \neq \emptyset$ then
            $P_{mal} := P_{mal} \cup \{p\}$
        ▷ Reconstruct randomness $r'$ of $p$
        $r' := \textsc{SumDC}[p] - \textsc{DC-Slot-Pad}(P', p, \mathrm{K}'[\,], \mathtt{sum})$
        ▷ Verify that the balance of $p$ is correct
        if $\mathrm{C}[p] = \left(\bigoplus_{(c,\pi,addr)\in Out'} c\right) \oplus \mathsf{Com}(fee/|P|, -r')$ then
            $P_{mal} := P_{mal} \cup \{p\}$
        ▷ Verify that $p$ has published correct symmetric keys
        for all $p_{ex} \in P_{ex}$ do
            if $\mathrm{K}_{ex}[p][p_{ex}] \neq \mathrm{K}'[p_{ex}]$ then
                $P_{mal} := P_{mal} \cup \{p\}$
    return $P_{mal}$
procedure $\textsc{Result-Of-Next-Run}(P_{exNext}, next)$
    ▷ Hand over to next run and notify of users to exclude
    notify $next$ of $P_{exNext}$
    ▷ Return result of next run
    $result :=$ join $next$
    return $result$
procedure $\textsc{VerifyResult}(i, P, Out, \mathrm{C}[\,], r_{\Delta})$
    $Out_{mal} := \emptyset$
    ▷ Verify range proofs
    for all $out \in Out$ do
        $(c, \pi, addr) := out$
        if $\mathsf{RPVerify}(\pi, c)$ then
            $Out_{mal} := Out_{mal} \cup \{out\}$
    ▷ Check balance
    $balanced := \left(\bigoplus_{p\in P} \mathrm{C}[p] = \left(\bigoplus_{(c,\pi,addr)\in Out} c\right) \oplus \mathsf{Com}(fee, -r_{\Delta})\right)$
    return $(balanced, Out_{mal})$
procedure $\textsc{GenOutputs}(Payments)$
    $myr := 0$
    $myOut := \emptyset$
    for all $(recipient, amount) \in Payments$ do
        $r \xleftarrow{\$} \mathcal{R}$                ▷ Fresh random value; implicitly stored in wallet
        $c := \mathsf{Com}(amount, r)$
        $\pi := \mathsf{RPCreate}(amount, r)$
        $myr := myr + r$
        $addr :=$ FreshRecipientAddress$(recipient)$
        $myOut := myOut \cup \{(c, \pi, addr)\}$
    **return** $(myr, myOut)$
**procedure** ConfirmTx$(i, P, my, \mathrm{VK}[\,], sk, \mathrm{UTXO}[\,], Out, sid)$
    $tx :=$ CoinJoinTx$(\mathrm{UTXO}[\,], Out)$
    $\sigma[my] :=$ Sign$(sk, tx)$
    **broadcast** $\sigma[my]$
    **receive** $\sigma[\,]$ **from all** $p \in P$
        **where** Verify$(\mathrm{VK}[p], \sigma[p], tx)$
    **missing** $P_{off}$ **do**                    ▷ Users refusing to sign are malicious
        **return** $P_{off}$
    Submit$(tx, \sigma[\,])$
    **return** $\emptyset$                                    ▷ Success!
```

A.1 Security Analysis

We argue briefly why ValueShuffle achieves the desired security and privacy properties.

Unlinkability. Unlinkability follows from sender anonymity in DiceMix [34]: Whenever some honest user i signs the CoinJoin transaction, the confirmation phase has been reached. In this case, because output triples are freshly generated for each run, DiceMix guarantees that the honest users form a proper DC-net. This in turn ensures that the attacker cannot distinguish whether an output triple of user i belongs to i or some other honest user j. Note that the relation between user and output triple can be revealed in the blame phase, but then a CoinJoin transaction with the current output triple will never be signed, so it is safe to reveal the relation. Instead, the output triple will be discarded and a further run will be started, using a fresh output triple, which is unlinkable to the discarded output triples. We refer the reader to DiceMix [34] for a detailed discussion. The only difference from DiceMix is that ValueShuffle runs two proper DC-nets in parallel.

CT Compatibility. ValueShuffle should not impair the privacy guarantees provided by CT. The only secrets of CT belonging to user i that ValueShuffle uses (and actually reveals in the blame phase) is her value $r - \sum_k r'_k$, where k ranges over her output triples. However, since r and all r'_k are random, this sum does not reveal anything about the individual r and r'_k values and thus does not affect the hiding property of the input commitment or any of the individual output commitments.

Termination. DiceMix itself provides termination, and we have to argue that our extensions do not affect this property. This mainly boils down to ensuring that a malicious user can be detected in each protocol run.

If one of the DC-nets for mixing the output triples is disrupted, then a malicious user will be identified. This follows from the termination of DiceMix and the observation that each message chunk is unpredictable. If the DC-net for computing r_Δ is disrupted, then the blame phase will sacrifice anonymity for this run (discarding the output triples), and the malicious user will be identified by checking her individual balance property, i.e., whether just her set of inputs and outputs are balanced, as done in the blame phase. If a wrong range proof is provided, then the blame phase will sacrifice anonymity for this run (discarding the output triples), and the malicious user can be identified by checking who provided the wrong range proof. In all other cases, the transaction will be valid by construction, so users refusing to sign it are malicious (or offline) and thus can be excluded from further runs.

By construction, and if the bulletin board is honest (which we assume for termination, as otherwise it is impossible to achieve), all honest users agree on the set of users to exclude and thus on the set of remaining users in the subsequential run of the protocol. We refer the reader to DiceMix [34] for details of termination.

Theft Resistance. The protocol must ensure that no honest user incurs money loss (ignoring transaction fees). ValueShuffle ensures theft resistance since the mixing of output triples and randomness does not involve the transfer of funds. Before the CoinJoin transaction is formed, every honest user checks that her output address and the corresponding committed value is included and only then signs the transaction. As a CoinJoin transaction becomes valid only when every user has signed the transaction (and thus confirmed that her funds are not stolen), ValueShuffle provides theft resistance.

References

[1] Andresen, G.: Pay to script hash, BIP 16. `https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki`

[2] Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in Bitcoin. In: FC'13

[3] Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better. how to make Bitcoin a better currency. In: FC'12

[4] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: S&P'14

[5] Bissias, G., Ozisik, A.P., Levine, B.N., Liberatore, M.: Sybil-resistant mixing for Bitcoin. In: WPES'14

[6] Bitcoin Core: Segregated witness: the next steps, `https://bitcoincore.org/en/2016/06/24/segwit-next-steps/#schnorr-signatures`

[7] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J., Felten, E.: Mixcoin: Anonymity for Bitcoin with accountable mixes. In: FC'14

[8] Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. J. Cryptol. 22(4) (2009)

[9] Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. J. Cryptol. 1(1) (1988)

[10] Corrigan-Gibbs, H., Ford, B.: Dissent: Accountable anonymous group messaging. In: CCS'10

[11] Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: USENIX Security'04

[12] Elements Project: Alpha sidechain, `https://www.elementsproject.org/sidechains/alpha/`

[13] Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: PKC'13

[14] Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC'11

[15] Gibson, A.: An investigation into Confidential Transactions (2016), `http://diyhpl.us/~bryan/papers2/bitcoin/An%20investigation%20into%20Confidential%20Transactions%20-%20Adam%20Gibson%20-%202016.pdf`

[16] Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub. In: NDSS'17

[17] Heilman, E., Baldimtsi, F., Goldberg, S.: Blindly signed contracts: Anonymous on-blockchain and off-blockchain Bitcoin transactions. In: BITCOIN'16

[18] Jedusor, T.E.: Mimblewimble, `https://scalingbitcoin.org/papers/mimblewimble.txt`

[19] Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in Bitcoin using P2P network traffic. In: FC'14 (2014)

[20] Lombrozo, E., Lau, J., Wuille, P.: Segregated witness (consensus layer), BIP 141. `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki#p2wpkh-nested-in-bip16-p2sh`

[21] Maxwell, G.: CoinJoin: Bitcoin privacy for the real world. Post on Bitcoin Forum (2013), `https://bitcointalk.org/index.php?topic=279249`

[22] Maxwell, G.: Confidential transactions (2015), `https://people.xiph.org/~greg/confidential_values.txt`

[23] Maxwell, G., Poelstra, A.: Borromean ring signatures (2015), `https://github.com/Blockstream/borromean_paper/raw/master/borromean_draft_0.01_9ade1e49.pdf`

[24] Meiklejohn, S., Orlandi, C.: Privacy-enhancing overlays in Bitcoin. In: BITCOIN'15

[25] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: Characterizing payments among men with no names. In: IMC'13

[26] Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from Bitcoin. In: S&P'13

[27] Noether, S., Mackenzie, A.: Ring confidential transactions. Ledger (2016)

[28] Noether, S.: Review of CryptoNote white paper, `https://downloads.getmonero.org/whitepaper_review.pdf`

[29] OmegaStarScream: Bitcoin Core & pruning mode. Bitcoin Forum, `https://bitcointalk.org/index.php?topic=1599458.0`

[30] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO'91

[31] Poelstra, A.: Mimblewimble, `http://diyhpl.us/~bryan/papers2/bitcoin/mimblewimble-andytoshi-INCOMPLETE-DRAFT-2016-10-06-001.pdf`

[32] Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: SXSW'13

[33] Ruffing, T., Moreno-Sanchez, P., Kate, A.: CoinShuffle: Practical decentralized coin mixing for Bitcoin. In: ESORICS'14

[34] Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2P mixing and unlinkable Bitcoin transactions. In: NDSS'17

[35] van Saberhagen, N.: CryptoNote (2013), https://cryptonote.org/whitepaper.pdf

[36] Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. 4(3) (1991)

[37] Spagnuolo, M., Maggi, F., Zanero, S.: BitIodine: Extracting intelligence from the Bitcoin network. In: FC'14

[38] Todd, P.: Stealth addresses. Post on Bitcoin development mailing list, https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg03613.html

[39] Valenta, L., Rowan, B.: Blindcoin: Blinded, accountable mixes for Bitcoin. In: BITCOIN'15

[40] Wuille, P.: Hierarchical deterministic wallets, BIP 32. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[41] Wuille, P.: Schnorr-SHA256 module in libsecp256k1, https://github.com/sipa/secp256k1/blob/968e2f415a5e764d159ee03e95815ea11460854e/src/modules/schnorr/schnorr.md

[42] Ziegeldorf, J.H., Grossmann, F., Henze, M., Inden, N., Wehrle, K.: CoinParty: Secure multi-party mixing of bitcoins. In: CODASPY'15