

Mixing Confidential Transactions: Comprehensive Transaction Privacy for Bitcoin^{*}

Tim Ruffing¹ and Pedro Moreno-Sanchez²

¹Saarland University
`tim.ruffing@mmci.uni-saarland.de`

²Purdue 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 make Bitcoin more private. 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 proposal to hide payment values in transactions. ValueShuffle ensures a mixing participant’s anonymity and the confidentiality of her payment values not only against an attacker observing the blockchain but also against the other possibly malicious mixing participants and against network attackers. By combining ValueShuffle with Confidential Transactions and additionally Stealth Addresses, our solution provides *comprehensive privacy* (payer anonymity, payee anonymity, and payment value privacy) without breaking with the design or the features of the current Bitcoin system. We demonstrate that the combination of these three privacy-enhancing technologies creates synergies that overcome the two major obstacles which so far have prohibited the deployment of coin mixing in practice, namely that users need to mix funds of the same value, and need to do so before they can actually spend the funds. As a result, our approach unleashes the full potential of coin mixing as a privacy solution for Bitcoin.

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 [26, 5, 39, 22, 33, 4, 25] by showing many different privacy

^{*} ^{*} This is a draft (revision 2016-12-23); the most recent revision is available at <https://people.mmci.uni-saarland.de/~truffing/papers/valueshuffle.pdf>.

weaknesses with the current Bitcoin protocol. This state of affairs has led to a plethora of privacy-enhancing technologies [27, 6, 37, 9, 40, 20, 19, 42, 7, 5, 34] aiming at overcoming these shortcomings while without breaking with the fundamental design of the Bitcoin system.

Yet, current available solutions offer only partial solutions, focusing typically just on one aspect of privacy. For instance, Confidential Transactions (CT) [18] defines a transaction format that hides the payment value, and Stealth Addresses (SA) [16] defines a mechanism for payers to generate unique one-time addresses, which improves payee anonymity.

The most prevalent approach to ensure payer anonymity while keeping 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 [24] transaction, which enables the users to atomically transfer their money 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 money can easily be avoided. Additionally, if users exchange their output accounts by means of an anonymous broadcast protocol [11, 36, 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.

It is obviously highly desirable to combine all of these privacy-enhancing technologies but this approach 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 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 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 CoinShuffle++ [34], the most efficient P2P coin mixing protocol in the literature, which is based on the DiceMix paradigm [34]. Since ValueShuffle successfully combines coin mixing, SA and CT, the resulting currency provides comprehensive privacy, i.e., unlinkability, payee anonymity and value privacy. Based on 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.

1.2 Exploiting Synergies

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

First, all forms of coin mixing are traditionally heavily restricted to mixing funds of the same amount, because otherwise it is trivial for an observer to link inputs and outputs together just based on their monetary amount, no matter 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 plain.

Second, current P2P coin mixing protocols [35] 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 users can spend the now mixed funds to a recipient (in a normal transaction). 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 the additional CoinJoin transactions. State-of-the art centralized mixing solutions share this limitation [19]. As a result, privacy comes with an 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 coins directly to the expected receivers in the CoinJoin transaction, which is arguably the mode of use for which CoinJoin was designed.

1.3 Features of ValueShuffle

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

Comprehensive Privacy. This combination 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 known addresses (payee anonymity), and CT provides value privacy.

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

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 DiceMix protocol [34], it terminates in only $4 + 2f$ communication rounds in the presence of f disruptive users. That is, an uninterrupted run of ValueShuffle completes in four communication rounds.

No anonymous channel 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 [13]. (However, to avoid that an observer is 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 transactions outputs that use complex scripts, e.g., advanced smart contracts, and provides meaningful privacy guarantees for them.

Reduced fees and space requirements. Unlike ring signatures as for example deployed in Monero [29] that require to add a signatures with 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.

Taking this one step further, CoinJoin makes Bitcoin in fact more efficient assuming the availability of Schnorr signatures [38], which are planned to be deployed by Bitcoin Core in the future [8]. The introduction of Schnorr signatures will enable aggregate signatures using a interactive two-round protocol among the users in a CoinJoin transaction [2]. This protocol can easily be integrated in ValueShuffle, and since we can exploit parallelism, 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 saving in terms of blockchain space and verification time compared to individual transactions, and hence also reduces fees compared to individual 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 usage of privacy-enhancing technologies in Bitcoin.

Compatibility with pruning. Unlike in Zerocash [6, 1] or in Monero [29], using CoinJoin it can be publicly observed which transaction outputs are unspent. While this releases some information to the public, it allows to prune spent outputs from the set of (potentially) unspent transaction outputs. This helps to mitigate the scaling problems in 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

Coin mixing. A variety of coin mixing protocols have been proposed so far in the literature, based on different paradigms. CoinShuffle [36] and its successor CoinShuffle++ [34] use P2P mixing to create a CoinJoin [24] transaction. This approach has the advantage that theft is excluded trivially by the use of CoinJoin; however effort is required to ensure that the protocol terminates in the presence of malicious users. ValueShuffle is an extension of CoinShuffle++; both scale easily up to anonymity sets of moderate size (e.g., 50 users).

Another line of research defines mixing uses an intermediary tumbler [9, 40, 20, 19]. 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 payment using a TumbleBit needs at least two sequential Bitcoin transaction (unless one is willing to use payment channels). ValueShuffle needs only one transaction to perform a payment.

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. In ValueShuffle instead, values are hidden and variable denominations are supported.

CryptoNote. The CryptoNote [37] design (as for instance implemented in Monero [29]) 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 [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 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 [32]. Pruning reduces the storage requirements of nodes drastically by deleting old blocks and spent transaction once verified. This is impossible in CryptoNote because anonymity ensures that it is not entirely clear whether a transaction has been spent or not. A CoinJoin-based approach such as ValueShuffle does not suffer from 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 [21, 30] 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, even though very basic contracts (e.g., multi-signatures and simple payment channels) can be constructed, Mimblewimble is not compatible with more sophisticated smart contracts. In contrast, ValueShuffle seamlessly supports scripts as currently implemented in Bitcoin.

Zerocoin and Zerocash. Zerocoin [27] and its follow-up work Zerocash [6], whose implementation Zcash has recently been deployed [1], are crypto-currencies 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 assumptions [31] 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 Zcash.

3 Building Blocks

Peer-to-peer Mixing. A peer-to-peer (P2P) mixing protocol [11, 36, 34] allows a set of untrusted users to simultaneously broadcast their messages without requiring any trusted third party. This protocol ensures *sender anonymity*, that is, an attacker controlling the network and some of the participating users cannot associate a message to its corresponding honest user. In this work, we use DiceMix [34] due to best performance in practice; in particular finishes in only 4 communication rounds when undisrupted.

DoS resistance. Disruptive users, whose goal is to prevent honest users from mixing, can be exposed and excluded in DiceMix. To this end, users broadcast the ephemeral secret material used in a failed protocol run and at least one misbehaving user is then identified by replying the expected computations. This allows the rest of users to start a new run excluding the misbehaving one. Repeating this exclusion mechanism at most once for malicious user ensures that eventually only honest users participate in a run of the protocol and it terminates.

Freshness of messages. To guarantee anonymity, DiceMix requires mixed messages to be fresh [34] and have sufficient entropy.¹ Freshness enables DiceMix to sacrifice anonymity for failed runs in order to identify disruptive users. Since the messages will be discarded, and no transaction will be performed, this is not a problem. However, freshness is required to ensure that messages from a failed run cannot be linked to messages from the final successful run.

¹ This is in fact a requirement of all P2P mixing protocols ensuring anonymity and termination [34].

Confidential Transactions. Confidential Transactions (CT) [18] is a cryptographic protocol that allows a single user to perform a transaction transferring bitcoins such that none of the monetary values in the inputs or outputs is revealed (*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 commitments additively homomorphic in message and randomness, i.e., $\text{Com}(x, r) \oplus \text{Com}(x', r') = \text{Com}(x + x', r + r')$. As an example, assume a user has two input values x_1 and x_2 and two output values x_3 and x_4 . She can commit to x_1, x_2 and x_3 as $c_i := \text{Com}(x_i, r_i)$, where r_i is chosen uniformly at random. Then, she encodes x_4 as $c_4 := \text{Com}(x_4, r_4)$, where $r_4 = (r_1 + r_2 - r_3)$. A verifier can then verify the balance property by checking whether $c_1 \oplus c_2 \stackrel{?}{=} c_3 \oplus c_4$ (ignoring fees). In practice, CT uses Pedersen commitments and range proofs based on borrowmean ring signatures [23].

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

Monetary amounts are in fact represented not as integers but as floating point numbers with a public exponent and only the mantissa hidden in the commitment; this is to support large amounts efficiently. However throughout this paper, 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.

We refer the reader to [18, 17] for details of confidential transactions.

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.

- The payee can post a stealth address, which enables any payer to derive fresh addresses.²
- The payee can send a BIP32 public key [41] to the payer, which enables the payer to derive fresh addresses. The derivation index can be derived from public information, e.g., a hash of the value commitment. (The payment protocol can be used to send the BIP32 public key.)
- 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

² CT and SA both rely on a key exchange between payer and payee. When combined together, the key exchange for CT can further be used for the stealth address as well (deriving two distinct independent keys from the shared secret), which effectively saves the key exchange for the stealth address.

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 cannot 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 describe ValueShuffle, the first P2P coin mixing protocol compatible with CT.

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 a tailored version of DiceMix, we rely on the same communication model. For completeness, we sketch this model here and refer the reader to [35] for details. We assume that users communicate to each other via a bulletin board, i.e., a server echoing messages from one user to the rest, e.g., an IRC server. 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 matter of communication.

4.1 Security and Privacy Features

ValueShuffle provides the following security and privacy guarantees.

Unlinkability: Given an output in the transaction, the attacker is not able to tell which honest user pays it.

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

Theft resistance: Funds of honest user either are 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. (Otherwise, all communication could be blocked by the attacker and liveness is impossible to ensure.)

4.2 Challenges and Our Solutions

To combine coin mixing with CT and unique 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 transactions, 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 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. In our case, the confirmation mechanism 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 [2].

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 straight-forward: A user that refuses to sign the final CoinJoin transaction, or provides wrong signatures (or wrong partial signatures in 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 [35] 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 recipient, 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 unique addresses and CT, we are able to guarantee anonymity while at the same time avoiding pre-mixing by performing ordinary transactions in the mixing. In contrast, previous coin mixing protocols (such as CoinShuffle++ [35]) 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 is desired [35].

Multiple-payer CT. While in the original design of CT, the single payer can easily create a crafted randomness for the commitments to input and output values in the transaction such that anyone can verify its correctness (see Section 3). However, in a mixing transaction with several payers, a naive construction of such verifiable transaction would require that users reveal to each other the randomnesses used in the commitments, thereby compromising the hiding property of the commitments in the first place.

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. As a sum, r_Δ does not reveal any information from individual random values used by users. Now, the overall CT becomes verifiable by checking whether $\bigoplus c_i - \bigoplus c'_i \stackrel{?}{=} \text{Com}(0, r_\Delta)$.

The value r_Δ can be obtained with a standard secure sum protocol [14] based on secret sharing, where every user i broadcasts her value $r_i - r'_i$ blinded by multiple keys, 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 a DC-net, where users use addition instead of XOR to mask their input messages.

Combining P2P mixing and secure sum. Since both DiceMix and our 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 one key exchange and derive independent subkeys for the P2P mixing and the P2P 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.

Handling disruption in secure sum protocol. Disruptive users cannot only disrupt the mixing of output triples but also the secure sum protocol by creating an output value commitments 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 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 reveal also her intended transaction value.

Mixing long messages in DiceMix. While DiceMix in its current form is practical for small messages m (e.g., $|m| = 160$ bit), it is prohibitively slow for messages of the size we require; we need $|m| \approx 20000$ bit to mix the quite large range proofs necessary used in CT. The most expensive step is polynomial factorization and requires each message to be an element of a finite field and consequently the finite field must have size of about $2^{|m|}$.

To overcome this issue, we split m into several chunks of fixed size, i.e., $m = m_1 || \dots || m_\ell$ 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 consists in prefixing every m_i for $1 < i \leq \ell$ with $H(m_1)$ so that every user mixes: $m'_1 = m_1$ and $m'_i = H(m_1) || m_i$

for $1 < i \leq \ell$. This arrangement allows for a trade-off between the computation and communication required for mixing: bigger chunks reduce the number of parallel mixing instances required but demands higher computation cost for polynomial factorization.

Supporting arbitrary scripts. So far we have talked only about output addresses but not about their type. (Note that for efficiency, an implementation will mix the binary hash and not its base64 encoding, without the checksum.) While mixing works fine with ordinary pay-to-pubkey-hash (P2PKH) hashes, we require pay-to-script-hash (P2SH) hashes [3]³ to support arbitrary scripts. However, it is not straightforward to mix P2PKH hash and P2SH hashes in the same mixing, because this would require to add the address type explicitly to the mixing message, which breaks anonymity: in case of a disruption, it becomes clear which inputs go to a P2PKH address and which inputs go to a P2SH address. To support P2PKH and P2SH together, we can instead perform P2PKH transactions via P2PKH. This creates a small overhead in the blockchain when spending (because the spender must show the script as a witness). If desired, this small overhead can easily be avoided a small soft-forking change in Bitcoin that introduces a single public key as a special P2SH type. Then the corresponding address will not reveal the type (P2PKH or P2SH), and even the responsibility of showing the type is moved to the spender.

4.3 Overview of ValueShuffle

We assume that every user i is represented by a triple $in_i = (c_i = \text{Com}(x_i, r_i), \pi_i, vk_i)$, 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. However, ValueShuffle can easily be adapted to overcome these assumptions.

From a high-level perspective, an execution of ValueShuffle may consists of several *runs*, and an individual run of ValueShuffle can be divided in four phases.

1. *Output generation.* Every user i locally generates her output triple $out_i = (c'_i = \text{Com}(x'_i, r'_i), \pi'_i, addr'_i)$, 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 anonymously 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, replacing one well-defined output commitment c'_j (e.g., the lexicographically smallest) by $c''_j := c'_j \oplus \text{Com}(0, -r_\Delta)$; note that j is unknown to the users due to anonymity.

³ 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 showing the script.

3. *Check.* Before users proceed, they 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 $\bigoplus_i c_i = \bigoplus_i c'_i$. 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 only checked 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 previous checks fail, this phase is reached to detect at least one misbehaving user. For that, every user i broadcasts the secrets she used for the mixing and secure sum protocols, thereby revealing the value $r_i - r'_i$ used by each user i . It is interesting to note here that users do not need to reveal the individual randomness as it would clearly reveal the payment value. Instead, the value $r_i - r'_i$ is enough to check that user i committed the same value in the input and output addresses (and therefore no coins were created).

In general, every other user j can recompute the mixing and secure sum steps from user i and detect whether she faithfully followed the protocol specification. The thereby found misbehaving 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). We refer to [35] for details. 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.

Overall, the wall-clock execution time of ValueShuffle will be dominated by communication time as in DiceMix. While our messages are longer than those used in the performance evaluation of DiceMix, the increase in the communication and computation is within reasonable bounds, and we expect ValueShuffle to scale to anonymity sets of at least moderate size (e.g., ≈ 50 users) in practice.

4.5 Protocol Description

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

Digital Signatures. We require a digital signature scheme (**KeyGen**, **Sign**, **Verify**) unforgeable under chosen-message attacks (UF-CMA). The algorithm **KeyGen** returns a private signing key sk and the corresponding public verification key vk . On input message m , **Sign**(sk, m) returns σ , a signature on message m using

signing key sk . The verification algorithm $\text{Verify}(pk, \sigma, m)$ 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 (NIKE.KeyGen , NIKE.SharedKey) secure in the CKS model [15, 10]. The algorithm $\text{NIKE.KeyGen}(id)$ outputs a public key npk and a secret key nsk for given a party identifier id . $\text{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid)$ outputs a shared key for the two parties and session identifier sid . 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 $\text{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid) = \text{NIKE.SharedKey}(id_2, id_1, nsk_2, npk_1, sid)$. Additionally, we require an algorithm $\text{NIKE.ValidatePK}(npk)$, which outputs *true* if and only if npk is a public key in the output space of NIKE.KeyGen , and we require an algorithm $\text{NIKE.ValidateKeys}(npk, nsk)$ which outputs *true* iff nsk is a secret key for the public key npk .

Static Diffie-Hellman key exchange satisfies these requirements [10], given a suitable key derivation algorithm such as $\text{NIKE.SharedKey}(id_1, id_2, x, g^y) := K((g^{xy}, \{id_1, id_2\}, sid))$ for a hash function K modeled as a random oracle.

Hash Functions. We require two hash functions H and G both modeled as a random oracle.

Confidential Transactions. Confidential Transactions (CT) relies on a non-interactive commitment scheme (Com , Open), which uses public parameters we keep implicit, and a range proof (RPCreate , RPVerify). Algorithm $c := \text{Com}(m, r)$ uses the randomness $r \in \mathcal{R}$ to output a commitment c of message m . Algorithm $b := \text{Open}(param, c, m, r)$ 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 $\text{Com}(m_1, r_1) \oplus \text{Com}(m_2, r_2) = \text{Com}(m_1 + m_2, r_1 + r_2)$. In practice, CT uses Pedersen commitments [12] as instantiation of a commitment scheme.

In a range proof scheme, the algorithm $\pi := \text{RPCreate}(m, r)$ creates a proof π of that $c = \text{Com}(m, r)$ is a commitment of a value in valid range. The algorithm $b := \text{RPVerify}(\pi, c)$ verifies that π is a valid range proof for c . We refer the reader to [18, 17] for details.

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

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

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

Regarding concurrency, a thread that runs a procedure $P(args)$ is started using “ $t := \mathbf{fork} P(args)$ ”, where t is a handle for the thread. A thread t can either be joined using “ $r := \mathbf{join} t$ ”, 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 balance can be adapted to consider the homomorphic combination of several input commitments.

As a result of these assumption, 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 $c = \mathbf{Com}(x, r)$, 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.

Full pseudocode. We describe the full protocol in pseudocode. We assume that the reader is familiar with the details of DiceMix [35] to understand the code. For better readability, the code for handling disruption (including offline users) and for parallel runs is blue. Our essential changes to DiceMix, which result in ValueShuffle, are in red.

```

1: procedure DCEMIX( $P, my, VK[], sk, UTXO[], C[], r, payments, sid$ )
2:    $sid := (sid, P, VK[])$ 
3:   if  $my \in P$  then
4:     fail “cannot run protocol with myself”
5:   return RUN( $P, my, VK[], sk, sid, 0$ )
6: procedure RUN( $P, my, VK[], sk, sid, run$ )
7:   if  $P = \emptyset$  then
8:     fail “no honest peers”
9:    $\triangleright$  Exchange pairwise keys
10:  ( $NPK[my], NSK[my]$ ) := NIKE.KeyGen( $my$ )

```

```

11:  $sidHpre := H((sidHpre, sid, run))$ 
12: broadcast  $(KE, NPK[my], Sign(sk, (NPK[my], sidHpre)))$ 
13: receive  $(KE, NPK[p], \sigma[p])$  from all  $p \in P$ 
14:   where  $NIKE.ValidatePK(NPK[p])$ 
       $\wedge Verify(VK[p], \sigma[p], (NPK[p], sidHpre))$ 
15: missing  $P_{off}$  do
16:    $P := P \setminus P_{off}$  ▷ Exclude offline users
17:    $sidH := H((sidH, sid, P \cup \{my\}, NPK[], run))$ 
18:    $K[] := DC-KEYS(P, NPK[], my, NSK[my], sidH)$ 
19:   ▷ Generate fresh outputs to mix
20:    $(myOut, myr) := GENOUTPUTTRIPLE(r, payments)$ 
21:    $DC[my][][[]] := DC-MIX(P, my, K[], Out)$ 
22:    $SUMDC[my] := myr + DC-SLOT-PAD(P, my, K[], sum)$ 
23:    $P_{ex} := \emptyset$  ▷ Malicious (or offline) users for later exclusion
24:   ▷ Commit to DC-net vector
25:    $COM[my] := H((CM, DC[my][], SUMDC[my]))$ 
26:   broadcast  $(CM, COM[my], Sign(sk, (COM[my], sidH)))$ 
27:   receive  $(CM, COM[p], \sigma[p])$  from all  $p \in P$ 
28:     where  $Verify(VK[p], \sigma[p], (COM[p], sidH))$ 
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:      $P_{exPrev} := \text{wait}$ 
34:      $P_{ex} := P_{ex} \cup P_{exPrev}$ 
35:     ▷ Collect shared keys with excluded users
36:     for all  $p \in P_{ex}$  do
37:        $K_{ex}[my][p] := K[p]$ 
38:     ▷ Start next run (in case this one fails)
39:      $P := P \setminus P_{ex}$ 
40:      $next := \text{fork RUN}(P, my, VK[], sk, sid, run + 1)$ 
41:     ▷ Open commitments and keys with excluded users
42:     broadcast  $(DC, DC[my][], SUMDC[my], K_{ex}[my][], Sign(sk, K_{ex}[my][[]]))$ 
43:     receive  $(DC, DC[p][], SUMDC[p], K_{ex}[p][], \sigma[p])$  from all  $p \in P$ 
44:     where  $H((CM, DC[p][], SUMDC[p])) = COM[p]$ 
       $\wedge \{p' : K_{ex}[p][p'] \neq \perp\} = P_{ex}$ 
       $\wedge Verify(VK[p], K_{ex}[p][], \sigma[p])$ 
45:   missing  $P_{off}$  do ▷ Abort and rely on next run
46:     return  $RESULT-OF-NEXT-RUN(P_{off}, next)$ 
47:    $Out := DC-MIX-RES(P \cup \{my\}, DC[my][], P_{ex}, K_{ex}[my][[]])$ 
48:   ▷ Patch one commitment such that values and randomness add up to zero
49:    $r_{\Delta} := DC-SLOT-OPEN(P_{all}, SUMDC[], sum, P_{ex}, K_{ex})$ 
50:    $(c, \pi, addr) := \min(Out)$  ▷ Lexicographically smallest output
51:    $c^* := c \oplus Com(0, -r_{\Delta})$ 
52:    $Out := (Out \setminus \{(c, \pi, addr)\}) \cup \{(c^*, \pi, addr)\}$ 
53:   ▷ Check if our output is contained in the result
54:    $(balanced, Out_{mal}) := VERIFYRESULT(i, P, Out, C[])$ 
55:   if  $myOut \subseteq Out \wedge balanced \wedge Out_{mal} = \emptyset$  then

```

```

56:    $P_{mal} := \text{CONFIRMTX}(i, P, Out, my, \text{VK}[], sk, \text{UTXO}[], sid)$ 
57:   if  $P_{mal} = \emptyset$  then ▷ Success?
58:     abort next
59:     return  $m$ 
60:   else
61:     broadcast  $(SK, NSK[my])$  ▷ Reveal secret key
62:     receive  $(SK, NSK[p])$  from all  $p \in P$ 
63:     where  $\text{NIKE.ValidateKeys}(\text{NPK}[p], NSK[p])$ 
64:     missing  $P_{off}$  do ▷ Abort and rely on next run
65:     return  $\text{RESULT-OF-NEXT-RUN}(P_{off}, next)$ 
66:     ▷ Determine malicious users using the secret keys
67:      $P_{mal} := \text{BLAME}(P, \text{NPK}[], my, NSK[], \text{DC}[[[]], sidH, P_{ex}, K_{ex}[[[]], Out_{mal}, C[]])$ 
68:     return  $\text{RESULT-OF-NEXT-RUN}(P_{mal}, next)$ 
69: procedure  $\text{DC-MIX}(P, my, K[], Out)$ 
70:   for  $o := 1, \dots, |Out|$  do
71:     ▷ Split message into chunks and prefix those
72:      $C[1] \parallel c := m$  ▷  $|C[1]| = \ell$ 
73:      $C[2] \parallel \dots \parallel C[n] := c$  ▷  $\forall j. |C[j]| = \ell - |H(C[1])|$ 
74:     for  $j := 2, \dots, n$  do
75:        $C[j] := H(C[1]) \parallel C[j]$ 
76:     ▷ Create power sums in individual slots
77:     for  $j := 1, \dots, n$  do
78:       for  $i := 1, \dots, |P| + 1$  do4
79:          $\text{DCMY}[o][j][i] := C[j]^i + \text{DC-SLOT-PAD}(P, my, K[], (o, j, i))$ 
80:     return  $\text{DCMY}[[[]][[]]$ 
81: procedure  $\text{DC-MIX-RES}(P_{all}, \text{DCMIX}[[[]][[]], P_{ex}, K_{ex}[[[]])$ 
82:   for  $j := 1, \dots, n$  do
83:     for  $s := 1, \dots, |P_{all}|$  do
84:        $M^*[s] := \text{DC-SLOT-OPEN}(P_{all}, \text{DCMIX}[[j][[]], s, P_{ex}, K_{ex}[[[]])$ 
85:       ▷ Solve equation system for array  $M[]$  of messages
86:        $M[j][i] := \text{Solve}(\forall s \in \{1, \dots, |P_{all}|\}. M^*[s] = \sum_{i=1}^{|P_{all}|} M[i]^s)$ 
87:       ▷ Recombine messages
88:        $M := \emptyset$ 
89:       for  $i := 1, \dots, |P_{all}|$  do
90:          $m := M[1][i]$ 
91:         for  $j := 2, \dots, n$  do
92:            $S := \{(i, m^*) : h \parallel m^* = M[j][i] \wedge h = H(M[1][i])\}$ 
93:           ▷ Unique match?
94:           if  $\exists i, m^*. S = \{(i, m^*)\}$  then
95:              $m := m \parallel m^*$ 
96:           else
97:             continue (outer loop) ▷ Invalid encoding, ignore message
98:            $M := M \cup \{m\}$ 
99:       return  $M$ 
100: procedure  $\text{DC-SLOT-PAD}(P, my, K[], s)$ 

```

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


```

101:   return  $\sum_{p \in P} \text{sgn}(my - p) \cdot G((K[p], s))$  ▷ in  $\mathbb{F}$ 
102: procedure DC-SLOT-OPEN( $P_{all}, DC[[]], s, P_{ex}, K_{ex}[[]]$ )
103:   ▷ Pads cancel out for honest users
104:    $m^* := \sum_{p \in P_{all}} DC[p][s]$  ▷ in  $\mathbb{F}$ 
105:   ▷ Remove pads for excluded users
106:    $m^* := m^* - \sum_{p \in P_{all}} DC-SLOT-PAD(P_{ex}, p, K_{ex}[[]], s)$ 
107:   return  $m^*$ 
108: procedure DC-KEYS( $P, NPK[[]], my, nsk, sidH$ )
109:   for all  $p \in P$  do
110:      $K[p] := \text{NIKE.SharedKey}(my, p, nsk, NPK[p], sidH)$ 
111:   return  $K[[]]$ 
112: procedure BLAME( $P, NPK[[]], my, NSK[[]], DC[[][][]], sidH, P_{ex}, K_{ex}[[]], Out_{mal}, C[[]]$ )
113:    $P_{mal} := \emptyset$ 
114:   for all  $p \in P$  do
115:      $P' := (P \cup \{my\} \cup P_{ex}) \setminus \{p\}$ 
116:      $K'[[]] := DC-KEYS(P', NPK[[]], p, NSK[p], sidH)$ 
117:     ▷ Reconstruct purported message  $m'$  of  $p$ 
118:     for  $o := 1, \dots, \text{len}(DC[my])$  do
119:        $m' := DC[p][o][1][1] - DC-SLOT-PAD(P', p, K'[[]], (o, 1, 1))$ 
120:       for  $j := 2, \dots, \text{len}(DC[p][o])$  do
121:          $m^* \parallel h := DC[p][o][j][1] - DC-SLOT-PAD(P', p, K'[[]], (o, j, 1))$ 
122:          $m' := m' \parallel m^*$ 
123:        $Out' := Out' \cup \{m'\}$ 
124:     ▷ Replay DC-net messages of  $p$ 
125:      $DC'[[][]] := DC-MIX(P', p, K'[[]], Out')$ 
126:     if  $DC'[[][]] \neq DC[p][[][]]$  then ▷ Exclude inconsistent  $p$ 
127:        $P_{mal} := P_{mal} \cup \{p\}$ 
128:     ▷ Verify that  $p$  has sent valid range proofs
129:     if  $Out' \cap Out_{mal} \neq \emptyset$  then
130:        $P_{mal} := P_{mal} \cup \{p\}$ 
131:     ▷ Reconstruct rerandomization factor  $r'$  of  $p$ 
132:      $r' := \text{SUMDC}[p] - DC-SLOT-PAD(P', p, K'[[]], \text{sum})$ 
133:     ▷ Verify that the balance of  $p$  is correct
134:     if  $C[p] = \left( \bigoplus_{(c, \cdot, \cdot) \in Out'} c \right) \oplus \text{Com}(fee/|P|, -r')$  then
135:        $P_{mal} := P_{mal} \cup \{p\}$ 
136:     ▷ Verify that  $p$  has published correct symmetric keys
137:     for all  $p_{ex} \in P_{ex}$  do
138:       if  $K_{ex}[p][p_{ex}] \neq K'[p_{ex}]$  then
139:          $P_{mal} := P_{mal} \cup \{p\}$ 
140:   return  $P_{mal}$ 
141: procedure RESULT-OF-NEXT-RUN( $P_{exNext}, next$ )
142:   ▷ Hand over to next run and notify of users to exclude
143:   notify  $next$  of  $P_{exNext}$ 
144:   ▷ Return result of next run
145:    $result := \text{join } next$ 
146:   return  $result$ 

```

```

147: procedure VERIFYRESULT( $i, P, Out, C[]$ )
148:    $Out_{incons} := \emptyset$ 
149:    $\triangleright$  Verify range proofs
150:   for all  $out \in Out$  do
151:      $(c, \pi, \cdot) := out$ 
152:     if RPVerify( $\pi, c$ ) then
153:        $Out_{incons} := Out_{mat} \cup \{out\}$ 
154:    $\triangleright$  Check balance
155:    $balanced := \left( \bigoplus_{p \in P} C[p] = \left( \bigoplus_{(c, \cdot, \cdot) \in Out} c \right) \oplus Com(fee, 0) \right)$ 
156:   return ( $balanced, Out_{incons}$ )
157: procedure GENOUTPUTTRIPLE( $r, payments$ )
158:    $myr := -r$ 
159:    $myOut := \emptyset$ 
160:   for all ( $recipient, amount$ )  $\in payments$  do
161:      $r' \xleftarrow{\$} \mathcal{R}$   $\triangleright$  Random value
162:      $c' := Com(amount, r')$ 
163:      $\pi' := RPCreate(amount, r')$ 
164:      $myr := myr + r'$ 
165:      $addr' := FreshRecipientAddress(recipient)$ 
166:      $myOut := myOut \cup \{(c', \pi', addr')\}$ 
167:   return ( $myr, myOut$ )
168: procedure CONFIRMTX( $i, P, my, VK[], sk, UTXO[], Out, sid$ )
169:    $tx := CoinJoinTx(UTXO[], Out)$ 
170:    $\sigma[my] := Sign(sk, tx)$ 
171:   broadcast  $\sigma[my]$ 
172:   receive  $\sigma[]$  from all  $p \in P$ 
173:     where Verify( $VK[p], \sigma[p], tx$ )
174:   missing  $P_{off}$  do  $\triangleright$  Users refusing to sign are malicious
175:     return  $P_{off}$ 
176:   Submit( $tx, \sigma[]$ )
177:   return  $\emptyset$   $\triangleright$  Success!

```

Note. Observe that the implementation of CONFIRMTX() is a simple transaction signed by several users. As noted above, alternative schemes are possible, e.g., the two-round aggregate Schnorr signature protocol [2]. 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 ConfirmTx effectively still takes only one round.

4.6 Security Analysis

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

Unlinkability. Unlinkability follows from sender anonymity in DiceMix [35]: 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 net.

This in turn ensures that the attacker cannot distinguish whether output triple of user i belongs to i or some other honest user j .⁵ (We refer to DiceMix [35] for a detailed discussion.) The only difference to 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 $r - \sum_i r'_i$ (where i ranges over her output triples). However, this value is just a re-randomization factor of a commitment and thus does not affect its hiding property.

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

If the DC-net mixing of output triples is disrupted, then a misbehaving user will be identified (this is exactly DiceMix’s termination property). 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 misbehaving user will be identified by checking her individual balance property, i.e., whether just her set of inputs and output is balance, 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 misbehaving 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 them 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), then the bulletin board performs broadcasts correctly, and 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 to DiceMix [35] for details of termination.

Theft Resistance. The protocol must ensure that no honest user incurs on 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 she 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.

⁵ 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.

5 Conclusion

In this work we describe ValueShuffle, the first time P2P mixing protocol that is fully compatible with Confidential Transactions, thereby providing comprehensive privacy guarantees. In doing so, our approach unleashes the full potential of CoinJoin-based P2P coin mixing in Bitcoin.

The additional practical features of ValueShuffle such as its compatibility with scripts, the reduction of required fees and space in the blockchain, its compatibility with pruning, and its layered design on top of the current Bitcoin system, make it a practical and efficient solution to ensure privacy and fungibility in Bitcoin.

Acknowledgements. This work was supported by the German Ministry for Education and Research (BMBF) through funding for the German Universities Excellence Initiative.

References

- [1] Zcash Website, <https://z.cash/>
- [2] Schnorr-sha256 module (2016), <https://github.com/sipa/secp256k1/blob/968e2f415a5e764d159ee03e95815ea11460854e/src/modules/schnorr/schnorr.md>
- [3] Andresen, G.: BIP 16: Pay to Script Hash, <https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki>
- [4] Androuraki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in bitcoin. In: FC'13
- [5] Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better. how to make Bitcoin a better currency. In: FC'12
- [6] 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
- [7] Bissias, G., Ozisik, A.P., Levine, B.N., Liberatore, M.: Sybil-resistant mixing for Bitcoin. In: WPES '14
- [8] Bitcoin Core: Segregated witness: the next steps, <https://bitcoincore.org/en/2016/06/24/segwit-next-steps/#schnorr-signatures>
- [9] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J., Felten, E.: Mixcoin: Anonymity for Bitcoin with accountable mixes. In: FC'14
- [10] Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. J. Cryptol. 22(4) (2009)
- [11] Corrigan-Gibbs, H., Ford, B.: Dissent: Accountable anonymous group messaging. In: CCS '10
- [12] Damgård, I., Pedersen, T.P., Pfitzmann, B.: Statistical secrecy and multibit commitments. IEEE Trans. Information Theory 44(3), 1143–1151 (1998), <http://dx.doi.org/10.1109/18.669255>
- [13] Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: USENIX Security'04
- [14] Franck, C., van de Graaf, J.: Dining cryptographers are practical. arXiv CoRR abs/1402.2269, <https://arxiv.org/abs/1402.2269> (2014)

- [15] Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: PKC'13 (2013)
- [16] genjix (pseudonym: Stealth address with SX (anonymous payments). Bitcoin Forum, <https://bitcointalk.org/index.php?topic=418071.0>
- [17] 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>
- [18] Greg Maxwell: Confidential Transactions (2015), https://people.xiph.org/~greg/confidential_values.txt
- [19] Heilman, E., Baldimtsi, F., Alshenibr, L., Scafuro, A., Goldberg, S.: TumbleBit: An untrusted tumbler for Bitcoin-compatible anonymous payments. IACR Cryptology ePrint 2016/575. <https://eprint.iacr.org/2016/575>
- [20] Heilman, E., Baldimtsi, F., Goldberg, S.: Blindly signed contracts: Anonymous on-blockchain and off-blockchain Bitcoin transactions. IACR Cryptology ePrint 2016/056. <https://eprint.iacr.org/2016/056>
- [21] Jedusor, T.E.: Mumblewimble, <https://scalingbitcoin.org/papers/mumblewimble.txt>
- [22] Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in Bitcoin using P2P network traffic. In: FC'14 (2014)
- [23] Maxwell, G., Poelstra, A.: Borromean Ring Signatures (2015), https://github.com/Blockstream/borromean_paper/raw/master/borromean_draft_0.01_9ade1e49.pdf
- [24] Maxwell, G.: CoinJoin: Bitcoin privacy for the real world. Post on Bitcoin Forum (2013), <https://bitcointalk.org/index.php?topic=279249>
- [25] Meiklejohn, S., Orlandi, C.: Privacy-enhancing overlays in Bitcoin. In: BITCOIN'15
- [26] 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
- [27] Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from Bitcoin. In: S&P'13
- [28] Noether, S.: Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098 (2015), <http://eprint.iacr.org/2015/1098>
- [29] Noether, S.: Review of Cryptonote White Paper, https://downloads.getmonero.org/whitepaper_review.pdf
- [30] Poelstra, A.: Mumblewimble, <http://diyhpl.us/~bryan/papers2/bitcoin/mumblewimble-andytoshi-INCOMPLETE-DRAFT-2016-10-06-001.pdf>
- [31] pg1989 (pseudonym): Non-falsifiable assumptions? Zcash Forum, <https://forum.z.cash/t/non-falsifiable-assumptions/64>
- [32] (pseudonym, O.: Bitcoin Core & pruning mode. Bitcoin Forum, <https://bitcointalk.org/index.php?topic=1599458.0>
- [33] Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: SXS'13
- [34] Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2P Mixing and Unlinkable Bitcoin Transactions. NDSS'17 (To appear), <http://eprint.iacr.org/2016/824>
- [35] Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2P Mixing and Unlinkable Bitcoin Transactions. NDSS'17 (To appear), <http://eprint.iacr.org/2016/824>
- [36] Ruffing, T., Moreno-Sanchez, P., Kate, A.: CoinShuffle: Practical decentralized coin mixing for Bitcoin. In: ESORICS'14 (2014)
- [37] van Saberhagen, N.: Cryptonote (2013), <https://cryptonote.org/whitepaper.pdf>

- [38] Schnorr, C.P.: Efficient signature generation by smart cards. *J. Cryptol.* 4(3), 161–174 (Jan 1991)
- [39] Spagnuolo, M., Maggi, F., Zanero, S.: BitIodine: Extracting intelligence from the Bitcoin network. In: *FC'14* (2014)
- [40] Valenta, L., Rowan, B.: Blindcoin: Blinded, accountable mixes for Bitcoin. In: *BITCOIN'15*
- [41] Wuille, P.: BIP32: Hierarchical Deterministic Wallets, <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>
- [42] Ziegeldorf, J.H., Grossmann, F., Henze, M., Inden, N., Wehrle, K.: CoinParty: Secure multi-party mixing of bitcoins. In: *CODASPY'15*