DMix: decentralized mixer for unlinkability

Conference Paper · September 2020

DOI: 10.1109/BRAINS49436.2020.9223282

2 authors:

Fadi Barbara
Università degli Studi di Torino
3 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Claudio Schifanella
Università degli Studi di Torino
73 PUBLICATIONS **867** CITATIONS
SEE PROFILE

DMix: decentralized mixer for unlinkability

Fadi Barbàra
Department of Computer Science
University of Turin
Torino, Italy
fadi.barbara@unito.it

Claudio Schifanella
Department of Computer Science
University of Turin
Torino, Italy
claudio.schifanella@unito.it

***Abstract*—We present a protocol that lets participants operate a decentralized mixer to exchange coins in the Bitcoin blockchain. DMix does not need the election of any leader and respects both the unlinkability and the atomicity properties, so that there is no possibility to correlate addresses or lose funds using the protocol. We leverage the MuSig aggregate signatures. This aggregation scheme is based on the Schnorr signature scheme, a recent proposal for a ECDSA alternative, the current Bitcoin signature scheme. We also present an analysis of the method and mitigation of attacks.**

***Index Terms*—blockchain, Schnorr signatures, MuSig, Bitcoin, decentralized mixing**

I. Introduction

Anonymous and privacy preserving digital currency has been a major topic in last years' cryptography literature. Recently, blockchain based currencies (often called *cryptocurrencies*) seem to enable these possibilities, but to date cryptographers have not achieved this goal entirely.

A consequence of this fact is that people can be tracked by analyzing the ledger. There are many heuristics which let analysts discover the real user behind a set of addresses [1], [2], [3], [4], [5], [6].

To solve these anonymity and privacy problems, blockchain communities are developing methods to contrast those heuristics. For example, in the Bitcoin community, Maxwell et al. developed CoinJoin [7] and its derivatives, while Heilman et al. [1] and Tram et al. [8] introduced mixers. CoinJoin allows users to aggregate inputs from different people to create a single transaction to multiple outputs. On the other hand, mixers (also called *tumblers*) let users anonymously exchange coins through an intermediary; if many users perform an exchange through the same intermediary, coins are effectively mixed, although the mixer operator can link old and new addresses together. Figure 1a illustrates the operation of a centralized mixing service. Both methods do not require any change to the Bitcoin block structure, while more complex proposals such as Mimblewimble [9] would require further developments given the substantial changes required in the Bitcoin protocol.

Analogous development is on-going in other permissionless blockchains. For example in the Ethereum [10] community the research is going toward more complex cryptographic schemes such as [11], which uses Secure Multiparty computation (MPC) and Trusted execution environments (TEEs) to achieve similar results, or zero-knowledge proof friendly mechanism as in recent developments in Ethereum 2.0 [12], [13]. These latter methods are still experimental and as of now they are in contrast with the "don't trust, verify" mindset: TEEs currently require a trusted third party[14] and being hardware based, it is difficult to promptly solve vulnerabilities [15]. On the other hand, current zero-knowledge non-interactive implementations like zk-SNARKS (used in ZCash) require a trusted setup to initialize the parameters and to date it is not proved that zero-knowledge proof based cryptocurrencies are immune from chain analysis attacks. For more details, see [16].

One of the new possibilities of recent proposed changes in the Bitcoin signature algorithms, from ECDSA to Schnorr signatures, is the creation of new protocols using aggregate signatures [17]. One of the advantages of an aggregate signature over multiple signatures (*multisignature*) is related to the privacy of the users. In fact, using a single signature regardless of the number of participants makes it nearly impossible to know if a certain address belongs to a single user or a multitude of users.

We propose to leverage aggregate signatures to create DMix, a decentralized mixer. Participants can control a jointly created (*aggregated*) address to send and then redistribute coins among them. This aggregate address acts as a mixer under full and exclusive control of the participants.

DMix leverages the Schnorr signatures which are not present in the Bitcoin protocol yet, but they are currently developed in the Bitcoin community and will be deployed soon. Schnorr signatures are needed to accomplish signature aggregation in a secure manner. Our protocol propose the use of MuSig [18], a multisignature scheme based on the Schnorr signature scheme. As far as we know, this is the first application of the scheme.

Thanks to the properties of the blockchain, our method does not require the election of any leader, and any party can control the whole process during all its phases. No party can cheat nonetheless: atomicity of the payment is preserved thanks to the use of the `CHECKLOCKTIME` opcode.

The paper proceeds as follows. In Section II we present

the relevant literature on the topic. In Section III we introduce the Schnorr and MuSig signature aggregation protocols and how we obtain atomicity in the protocol. In Section IV we present the method and in Section V we analyze it. In Section VI we introduce possible applications and we conclude the paper.

II. Related works

There are two current methods which address the traceability issue on Bitcoin : mixers and CoinJoin .

A. CoinShuffle

In CoinShuffle [19], the authors propose a method to mix coins in an anonymous and decentralized manner.

The authors assume that every participant holds the same amount of coins at some Bitcoin address and that this address will be one of the input addresses in the mixing transaction. Furthermore every message from this participant is signed with the private key associated with the public key/address.

The protocol is split into three phases. In the first one, participant create their new addresses, then they shuffle and broadcast them to write the transaction and in the last phase they verify the transactions. Particular care must be taken for the second phase: the shuffling.

If parties A,B and C want to perform a CoinShuffle, they have to decide an order of shuffling. Without loss of generality we can assume the order is lexicographical, and therefore it will be A,B and C. To perform the shuffle A (which acts as a leader) will encrypt its new address with C's public key first and then with B's one, using the non commutativity of encryption. When B receives the double encrypted message from A, he can decrypt it and see the encrypted (with C's public key) message. B will encrypt his new address with C's key, then B shuffles his message and the A's one and finally he sends both messages to C.

C can decrypt both messages and see the addresses. He can not say which new address belongs to A or B. He creates a transaction with those new addresses and broadcast it to the blockchain. This method presents a possible point of failure: if C does not broadcast the transaction, the whole process will stop.

B. TumbleBit

TumbleBit is a centralized mixer [1]. The authors propose a method to interactively exchange bitcoins between two parties in a way which promote unlinkability by creating two channels; one between the first user A and the Tumble T and the other between the other user B and the Tumble.

In the Payment phase, A pays T which then pays B. This can be scaled to hundreds of users, therefore providing anonymity thanks to a big anonymity set and the many possible choices of A and B.

The system provides unlinkability, balance [1] and security against DDoS and Sybil attacks. On the other hand, the system accomplish those goals being a trusted third party which could abuse its position (e.g. he can delay payments or refuse to establish channels with some user which won't be able to swap funds). Furthermore, to be able to provide a multitude of users with the required liquidity, the system must hold a great amount of bitcoins . This is an incentive for the creation of centralized payment hubs, similar to centralized exchanges.

C. CoinJoin

CoinJoin, originally proposed in [7], is a particular transaction which aggregates inputs of different people to jointly pay one or more parties. The goal is to build transactions in a way that tries to invalidate naive taint tracking. It does not need any change to the Bitcoin protocol and it is relatively easy to perform, but it needs interactive coordination between parties.

CoinJoins try to invalidate the heuristic that can be defined as follows [4]: If two (or more) addresses are inputs to the same transaction, they are controlled by the same user.

On the one hand, CoinJoins invalidate the efficacy of the heuristic [20], on the other hand users risk to come into contact with *dirty* coins, such as bitcoins suspected to come from a theft or illegal black market: the innocent person could be considered to be the author of such theft if the heuristic is applied and considered as valid. At this point he must prove that he made a CoinJoin, losing his privacy [21].

Another analysis by Maurer et al. [22] suggests that normal CoinJoins do not provide unlinkability. To obtain unlinkability it is necessary to consider other mechanism, such as mixers.

III. Preliminaries

A. Schnorr Signatures

The Schnorr signature scheme was introduced by Schnorr et al. [23] as a method of authentication in smart cards. This scheme is under consideration to replace the currently used ECDSA signature scheme in Bitcoin protocol[17]. The signature scheme change is part of a bigger soft-fork proposal which involves a total of three Bitcoin Improvement Proposals (BIPs): BIP340, BIP341 and BIP342. The first one is the BIP related to Schnorr Signatures.

Schnorr signatures present advantages with respect to the ECDSA signature scheme in the Bitcoin context. First, Schnorr signatures are provably secure with assumptions that are weaker than those required by ECDSA signature scheme:[24], [25], [26]. Moreover, the inherent non-malleability of Schnorr signatures solves the malleability of ECDSA in a stronger way. For more details see [17]

[1]Balance means that there is no possibility of inflation or destruction of money even if parties colludes

Finally, its linearity permits signature aggregation, i.e. the possibility of homomorphically sign a message using more than one key. In fact, given $\mathbf{sig}(m, sk_1)$ and $\mathbf{sig}(m, sk_2)$ the Schnorr signatures of message m with secret/private key sk_1 and sk_2 respectively, and given $sk_1 + sk_2$ the aggregated key from sk_1 and sk_2, then

$$\mathbf{sig}(m, sk_1) + \mathbf{sig}(m, sk_2) = \mathbf{sig}(m, sk_1 + sk_2)$$

We use this property to build DMix.

We report here the Elliptic Curve Schnorr signature implementation variant used in Bitcoin. For details, see [17], for the original description see [23]. In the rest of the paper we will use the notation introduced in [18]

Key Generation A user generates $sk \xleftarrow{\$} \mathbb{Z}$ which acts as private key, and then computes the corresponding public key $pk = g^{sk}$, where g represents the generation point of the elliptic curve and the operation g^{sk} represent the usual elliptic curve operation of a point g over itself iterated sk times.

Signing Given a message m, a user wanting to sign a message execute the following instructions:

1) $k \xleftarrow{\$} \mathbb{Z}$
2) $r = g^k$
3) $e = H(r||m)$ where H is a hash function and $||$ means concatenation
4) $s = k + sk \cdot e$

The signature is the couple (r, s). This variation is preferred because it allows batch signature verification.

Verifying Given a message m, the public key pk and the signature (r, s) a verifier computes $e_{ver} = H(r||m)$. The signature is valid if $g^s = r \cdot pk^e$

In fact: $g^s = g^{k+sk \cdot e} = g^k g^{sk \cdot e} = g^k g^{sk \cdot e_{ver}} = r \cdot pk^e$

B. MuSig

MuSig is a multisignature protocol [18] built upon the Schnorr signature scheme [23].

This method protects from rogue attacks [27] with no additional overhead and therefore it permits safe signature aggregation. A rogue attack happens when a subset of t malicious signers over a total of n, $1 \leq t < n$, can compute the public keys $pk_{n-t+1}, \ldots, pk_n$ as functions of the public keys of the honest users $pk_1, \ldots, pk_{n-t}$. This process allows them to produce forgeries for the whole set of public keys $\{pk_1, \ldots, pk_{n-t}, pk_{n-t+1}, \ldots, pk_n\}$. For example, a malicious user seeing the public key pk_1, can broadcast the public key $pk_2 - pk_1$. This way the *common* public key would be $pk_1 + (pk_2 - pk_1) = pk_2$. From now on this malicious user can create (single) signatures alone which will be considered and verified as multisignatures, effectively signing on behalf of the honest party.

The signature scheme is secure in the plain public key model. This model has been introduced by Bellare et al. in [27] meaning that it requires only that each signer has a (certified) public key. A consequence is that there is no need of a PKI or key setup.

We briefly describe here the three algorithms of the MuSig signature process, using the same notation of [18].

Given signers $X_1, \ldots, X_n$, let the group parameters be $(\mathbb{G}, p, g)$ where p is a k-bit integer, $\mathbb{G}$ is a cyclic group of order p, and g is a generator of $\mathbb{G}$.

For the protocol, the authors use three different hashing functions:

$$H_{com}, H_{agg}, H_{sig} : \{0,1\}^* \to \{0,1\}^l$$

where l is a security parameter.

Let $L = \{pk_1, \ldots, pk_n\}$ be a multiset[2] of public keys. When this multiset is given as input to a hash function, the authors assume it to be uniquely encoded first, e.g. using the lexicographical order.

Throughout the paper we maintain the multiplicative notation of the authors.

Key Generation Each signer i generates a random number, which acts as private key, $sk_i \xleftarrow{\$} \mathbb{Z}$ and then computes the corresponding public key $pk_i = g^{sk_i}$ (same as key generation in Schnorr signatures).

To create the aggregated key, each participant gathers the public keys of the other participants and create the multiset L. Each signer then is able to independently create the aggregate public key by computing for each participant i the hash $a_i = H_{agg}(L, pk_i)$ and then aggregating this computation obtaining

$$PK = \prod_{i=1}^{n} pk_i^{a_i}$$

Signing In the signing phase the participants create the aggregate signature of a message m. Each participant i randomly create $r_i \xleftarrow{\$} \mathbb{Z}_p$, and then computes $R_i = g^{r_i}$ and its commitment $t_i = H_{com}(R_i)$.

Each signer sends t_i and after he collected all the other commitments he sends R_i. Upon the receiving of all the other R_j, he checks that $t_j = H_{com}(R_j), \forall j$: if this is not the case, he aborts the protocol.

Assuming everything is fine, the i-th signer computes:

$$R = \prod_{i=1}^{n} R_i, \quad e = H_{sig}(pk, R, m), \quad s_i = r_i + e a_i sk_i \pmod p$$

and sends s_i to all other cosigners.

In the end, if the i-th signer receives $\{s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n\}$ from the other cosigners, he can compute $s = \prod_{i=1}^{n} s_i \pmod p$. The aggregate signature is $\sigma = (R, s)$.

Verification Given the multiset of public keys L, a message m, and the signature $\sigma = (R, s)$, the verifier computes $a_i \forall i \in \{1, \ldots, n\}, PK$ and e as in the signing phase, and accepts the signature if

$$g^s = R \prod_{i=1}^{n} pk_i = R \cdot pk^e$$

.

[2]Extension of the concept of set which admits the repetition of elements.

C. Bitcoin transactions and Atomicity

Every Bitcoin transaction has an output script where there are encoded the conditions for redeeming it. Most transactions are executed with predefined forms of output scripts, such as pay-to-public-key-hash (P2PKH) and pay-to-script-hash (P2SH). These forms are called *standard* output scripts. Other forms of output scripts, *non-standard* output scripts, are generally ignored by the majority of nodes for security reasons[3].

Thanks to BIP 13 [29], any output script can be converted to P2SH. This allows users to use non-standard output scripts by publishing it in the form of a P2SH transaction. In particular, the output script we use in this protocol uses an opcode described in BIP 65 which defines the `CHECKLOCKTIMEVERIFY` opcode. This opcode allows a transaction output to be made unspendable until some point in the future. See [30] for details and examples.

In DMix, we use a particular form of escrow, described in [30]. Assume that there are two participants in an instance of DMix, *Alice* and *Bob*, that want to send money to the aggregate address created with the procedure described in the previous subsection. Let `HpkA`=pk_A be the hash of the public key of *Alice* and `HpkAB`=pk_{AB} be the hash of the aggregate public key. Then the script[4]:

```
IF <nBlock> CHECKLOCKTIMEVERIFY DROP <HpkA>
↪  CHECKSIGVERIFY 1
ELSE 1 ENDIF <HpkAB> 1 CHECKMUSIG
```

gives *Alice* and *Bob* a chance to spend the money signing the transaction with their common private key before block `nBlock`. If the two participants do not reach an agreement before that block, then *Alice* can still spend her money operating only with her private key. Therefore *Alice* is protected in case *Bob* is malicious, and the exchange is atomic. We call this output script $Script1$.

To publish a P2SH Bitcoin transaction based on $Script1$, *Alice* create the hash of $Script1$, `Hs1`, and then publish the following outputScript[5]:

```
HASH160 <Hs1> EQUAL
```

We call this transaction $tx1$. Redeeming transaction $tx1$ requires a transaction $tx2$ with four inputs: the output script of the hash in transaction $tx1$ ($Script1$ in our example), the relative inputScript (the inputScript which would evaluate to `true` once concatenated with $Script1$ in our example), the set of output addresses of $tx2$ and the respective output amounts.

The inputScript for $tx2$ in the case of an outputScript analogous to $Script1$ is of the form $(\mathbf{sig}(H(tx2), sk_A), pk_A)$ assuming Alice alone signs $tx2$ and therefore redeems transaction $tx1$, or $(\mathbf{sig}(H(tx2), sk_{AB}), pk_{AB})$ in case both *Alice* and *Bob* sign this transaction. Function $\mathbf{sig}(H(m), sk)$ produce a signature of message m using the key sk.

[3]It is possible to execute DDoS attacks by placing output scripts that are too complicated to verify [28].

[4]We used the non existent opcode `CHECKMUSIG` to simulate the opcode of MuSig because there is not any accepted specific at the time of this writing.

[5]See BIP16 at https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki.

D. Decentralized Mixers Properties

Similar to the CoinShuffle protocol proposal [19] we require our method to respect some properties. We present them below with an explanation.

- **No Third Party**: there must not be any external party but the participants, so that the protocol can be used with no need of external services;
- **Compatibility**: the protocol must not require any change in the Bitcoin protocol (assuming the deployment of Schnorr signature);
- **No Mixing Fee**: there is no external cost in executing the protocol; the only fees payed by the participants are required from sending transactions;
- **Small Overhead**: there is no overhead for participants in using the protocol, but only the need of authenticated channels;
- **Efficiency**: in case of honest participants, the protocol does not require any major time-based overhead other that the time needed for the creation of blocks;
- **Unlinkability**: with a successful run of the DMix protocol, a blockchain analyst can not ascertain common ownership of input address of the `inDMix` transactions and output addresses of the `outDMix` transaction;
- **Verifiability**: every participant should be able to verify the transaction and no participant should be able to steal coins;
- **Atomicity**: either all parties receive their coins back; either in an unlinkable manner (i.e. the DMix protocol is successful), or in a linkable manner (i.e. there was a faulty communication or a malicious user).

IV. The DMix Protocol

DMix is a decentralized protocol that acts as a decentralized mixer for the Bitcoin protocol. It provides atomicity and privacy leveraging the aggregated signature of MuSig over the Schnorr signature scheme.

In our protocol, we assume N distinct parties met either in person or online. The protocol requires a distinct transaction from each participant toward an aggregated address, DM, using the MuSig key generation algorithm created by the participant themselves; we call this kind of transactions `inDMix` transactions. Then, to redeem the coins, the participants need to create an aggregate signature of exactly one transaction that starts from DM and goes to multiple new addresses belonging to the participants; we call this transaction `outDMix`. Figure 1b illustrate both kind of transactions. No intermediary is needed to perform all of these steps.

Our protocol is divided into three steps which can be referred to as:

1) Information and Public Keys exchange

2) `inDmix` Transactions
3) `outDmix` Transaction

In subsequent subsections we describe every phase in detail. We put a running example at the end of each phase. In the example *Alice*, *Bob* and *Carol* need to mix their coins and decide to use DMix.

A. Information and Public Keys exchange

We assume the parties create a shared secure channel to communicate between each other. This channel remains open throughout the whole protocol and participants will close it when the exchange is finished.

To setup a DMix exchange, participants need to agree on five inputs in order:

1) the total fees they will pay in the `outDMix` transaction
2) the timeout of the whole process
3) the amount of the outputs in the `outDMix` transaction
4) the $\texttt{amountIn}_i$, i.e. the amount that each participant P_i sends to DM in the `inDMix` transaction
5) the information required to jointly create DM.

Total Fees: In the third phase of the protocol, participants send a transaction from the aggregated DMix address DM, so they need to agree on the total fee `totalFee` which they intend to give to the miners. If participants agree on the total fees, they can independently create the `outDMix` transaction. `totalFee` has to be divided between the participants, e.g. equally among them. Given participants $P_1, P_2, \ldots, P_N$ which have to pay $\texttt{fee}_1, \texttt{fee}_2, \ldots, \texttt{fee}_N$ fees respectively, then $\sum_{i=1}^{N} \texttt{fee}_i = \texttt{totalFee}$.

Timeout: In similar fashion, users decide the block number (either relative or absolute) for the `nLocktime` field of the `inDMix` transaction. This field is used by the `CHECKLOCKTIMEVERIFY` opcode and it acts as a timer: after that period, if the protocol aborted for some reason, users can redeem their coins (Section III-C). Thanks to this, the DMix protocol ensures the atomicity.

`amountOut`: Users have to establish a common granularity value for each output amounts of the `outDMix` transaction. We call this parameter `amountOut`.

Having a common output amount is crucial to preserve the unlinkability requirement in the decentralized mixer: different transaction output amounts could lead to the linking of input and output amounts and lose all the privacy property of a privacy preserving method due to the subset sum problem[6]. This problem is present in some privacy preserving methods such as CoinJoin [22], and to avoid this problem we require user to agree on a common output and decide their input based on this parameter.

[6]See for example the Wikipedia article at https://en.wikipedia.org/wiki/Subset_sum

$\texttt{amountIn}_i$: Each participant has to declare the amount he wants to send to DM. Differently from CoinShuffle, users can put different amounts of coins in DMix. Declaring the input amount lets every other participant to independently build the `outDMix` transaction and prevents the creation of a central point of failure. The only constraint is that $\texttt{amountIn}_i - \texttt{fee}_i$ must be an integer multiple of `amountOut` for each participant.

A possible drawback of this method is the creation of many outputs risking the creation of dust coins, i.e. coins whose value is less than transaction fees and therefore they are not spendable. In practice, this is not a real problem because people agree on the fees before this step and decide `amountIn` taking into account future fees.

DMix Address Creation: Participants in DMix create the aggregate address from the aggregated public key generated by the MuSig signature scheme. In particular, each participant P_i sends in the channel his own public key pk_i . After he collects the other public keys, he creates the aggregate public key PK (see Section III-B). Each participant derives the same public key pk and from that key the participant can create the DMix address DM independently[7].

Example *Alice*, *Bob* and *Carol* create a shared secure channel, see the leftmost column of Figure 1b decide the fees they intend to pay to the miners for the `outDMix` transaction, and the timeout. In this case they decide that the fees will be a total of 0.015Ƀ to be payed equally among them. After this step, they decide for the `amountOut` which in this case is 0.25Ƀ.They see that the bitcoin blockchain has created block number 1000 few minutes ago, and they decide to put the timeout in about three hours. They decide to put an absolute block number: the timeout is decided to be block number 1018. Based on these parameters, Alice decides her $\texttt{amountIn} = 1.005$Ƀ, Bob's $\texttt{amountIn} = 0.505$Ƀ and Carol's $\texttt{amountIn} = 0.255$Ƀ.

Finally Alice, Bob and Carol share their public key with the others and create the address DM following the key generation algorithm of the MuSig protocol.

B. *`inDmix`* Transactions

After discussing the terms of the transaction in the previous step, each participant sends a transaction from his address to the aggregate address DM created in the previous phase. Exactly one output of this transaction must be equal to the `amountIn` previously declared.

These `inDMix` transactions have a construction analogous to transaction $tx1$, as explained in Section III-C. For each participant P_i we call $\texttt{inDMix}_i$ the P2SH transaction he sends to DM. The relative script encoded in the hash of the P2SH transaction sent by P_i will be addressed as $script_i$. This phase ends when all participants $\{P_1, \ldots, P_N\}$ have sent their `inDMix` transactions and all of them have been included in the blockchain.

[7]The address DM is the `base58` of the hash of the public key pk [31]

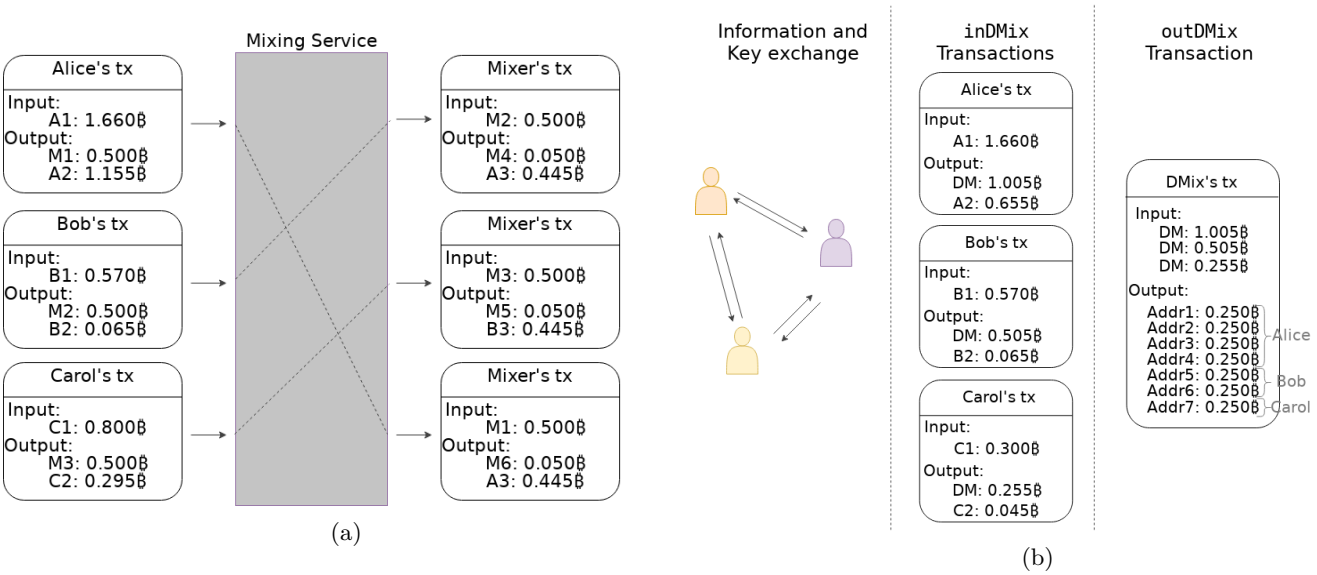

Fig. 1: A graphical comparison between a centralized mixing service and DMix. (a) A centralized mixer. The path of the mixing is known to the operators of the mixer. (b) DMix example. We marked Alice's, Bob's and Carol's addresses in gray because this information is known only to those who participated in the DMix protocol

Example If PK is the aggregated public key and DM is the aggregated address derived from it, we write `HPK`$=H(PK)$ the hash of PK. We call $A1$ the input address of *Alice* in the $\texttt{inDMix}_{Alice}$ transaction and $A2$ her change address. With `HpkA1`$=H(pk_{A1})$ the hash of *Alice*'s public key pk_{A1} relative to the address $A1$.

Alice's script for the P2SH transaction, denoted $script_{Alice}$, is:

```
IF <1018> CHECKLOCKTIMEVERIFY DROP DUP HASH160
↪ HpkA1 EQUALVERIFY ELSE DUP HASH160 HPK
↪ EQUALVERIFY ENDIF CHECKSIG
```

and its hash is denoted `HsA`. The $\texttt{inDMix}_{Alice}$ transaction of Alice has the following parameters:

- input Address: $A1$ output Address: DM, $A2$
- input Amount: 1.66฿
  output Amount: 1.005฿, 0.655฿
- inputScript[8]:`true`, outputScript:
  `HASH160 <HsA> EQUAL`

Bob and Carol send analogous transactions, see the middle column of Figure 1b.

C. *outDmix Transaction*

The goal of all participants in this phase is to redeem their coins. To accomplish this goal they need to spend the coins previously transferred to the DM address. In particular, the participants need to jointly create and sign a single transaction which spends all the inputs from the `inDMix` transactions, moving the coins to many outputs belonging to the participants: this is the `outDMix` transaction. The `outDMix` transaction is analogous to $tx2$ of Section III-C: participants have to create one such transaction and an aggregate signature of it to redeem their coins.

Of the four inputs needed to create this kind of transaction, participants already know the output amounts: these amounts are equal to `amountOut`, decided in the first phase. Furthermore, each participants already knows the aggregated public key: they only need to compute the aggregate signature to complete the input script of the `outDMix` transaction. For these reason these nine steps are needed in this phase. Each participant P_i:

1) sends his newly created addresses $\{Addr_1^i, \ldots, Addr_{m_i}^i\}$, where $m_i = \frac{\texttt{amountIn}_i - \texttt{fee}_i}{\texttt{amountOut}}$
2) sends $script_i$ used in the $\texttt{inDMix}_i$ transaction
3) checks the hashes for each $script_i$ received
4) sorts $\{Addr_1^i, \ldots, Addr_{m_i}^i\}, \forall i = 1 \ldots n$
5) assigns `amountOut` to each output amount and put $script_i$ in the right field
6) signs the `outDMix` transaction, obtaining sig_i which then shares with the other participants (Section III-B)
7) checks the received signatures using the verification algorithm of Section III-A
8) create the aggregate signature of the `outDMix` transaction (Section III-B)
9) sends the signed transaction

These steps give to each participant the ability to independently create and sign the same `outDMix` transaction that will be published to the blockchain. Miners will include only one of these and will treat the others as double-spend attempt, therefore discarding them. The advantage is that this way each participant is not trusting the others

[8]We put a single `true`, because we assume the user can spend a previously received output

anymore than he trusts himself. Therefore, thanks to the properties of the blockchain, they can redeem their money in a leader-less manner.

Once the transaction has been sent and parties received their funds, the users close the channel.

Example *Alice*, *Bob* and *Carol* communicate their newly created output addresses and their input scripts to redeem their coins. For easiness of explanations, let's say Alice's new addresses are $\{Addr1, Addr2, Addr3, Addr4\}$, Bob's addresses $\{Addr5, Addr6\}$ and Carol's address is $\{Addr7\}$. Assume these address are already in lexicographical order. *Alice* sends $script_{Alice}$, *Bob* sends $script_{Bob}$ and *Carol* sends $script_{Carol}$; *Alice* checks that $H(script_{Bob})$ =`HsB` from $\texttt{inDMix}_{Bob}$ and that $H(script_{Carol})$ =`HsC` from $\texttt{inDMix}_{Carol}$. Analogous things do both *Bob* and *Carol*.

The input addresses of their `outDMix` transaction are the output addresses of their respective $\texttt{inDMix}_i$, so the redeeming transaction will be:

- input: $DM : 1.005$Ƀ, $DM : 0.505$Ƀ, $DM : 0.255$Ƀ
- output: $Addr1 : 0.25$Ƀ, $Addr2 : 0.25$Ƀ, $Addr3 : 0.25$Ƀ, $Addr4 : 0.25$Ƀ, $Addr5 : 0.25$Ƀ, $Addr6 : 0.25$Ƀ, $Addr7 : 0.25$Ƀ

Alice, *Bob* and *Carol* obtain the same transaction independently and then they sign it, obtaining sig_{Alice}, sig_{Bob} and sig_{Carol}. Finally they collect the other signatures and independently aggregate them and publish their `outDMix` transaction. In the rightmost column of Figure 1b we represented this transaction.

V. Analysis

A. Properties satisfaction

We explain here how our protocol satisfies the properties previously listed. Obviously participants do not require any third party to cooperate (e.g. mail communication do not intrinsically need any third party) and there are no mixing fees; the only fees required are those belonging to the miners. Moreover, the protocol has a small overhead with respect to a normal transaction: it only requires one transaction to the DMix address and a transaction from it. For the same reason, the protocol is Efficient if the participants are honest. Assuming Schnorr signatures are deployed, the protocol is compatible with Bitcoin.

Atomicity derives from the the `inDMix` transaction built as in Section III-C. Similarly, Verifiability derives from the signature choices. In fact, every participant can check that the others are following the protocol: the protocol makes (honest) participants fill the required field of the `outDMix` transaction in a unique way; therefore by applying the verification algorithm of the Schnorr scheme to the collected signatures in the seventh step of the third phase (Section IV-C), the participants can be sure that they are creating a valid aggregate signature, i.e. a signature of the right transaction.

The last property is Unlinkability: thanks to the equivalence of all output amounts in the `outDMix` transaction and the fact that the communication between parties is private and/or encrypted, there is no possibility to link input and output in the transaction. Furthermore, this solve the issues of other privacy preserving methods such as CoinJoins (see Section IV-A)

B. Managing attacks

The method lets people exchange funds in an atomic and privacy preserving manner thanks to a private secure channel, aggregate signatures and the `CHECKLOCKTIMEVERIFY` opcode without third parties or any leader election. While we used techniques that prevent passive attacks, it is still possible to perform an internal active attack. In fact, a malicious participant could track all the addresses, linking them to the owners during the third phase and then broadcast the transcription of the communication.

To mitigate this attack, users can open private a channel of communication with other users. In this private channel, parties would privately exchange the addresses. In the common chat room those participants would communicate addresses not belonging to themselves, while being sure that others will communicate their own. We explain here the rationale from a probabilistic point of view.

Let I be the event {There is exactly one malicious user in the group} and set the probability of that event to p, i.e. $P(I) = p$. If there are N participants $\{P_1, \dots, P_N\}$, this means that I is the union of N different events:

$$I = \{P_1 \text{ is malicious}\} \cup \dots \cup \{P_N \text{ is malicious}\} \quad (1)$$

Because these are disjoint events, we assumed there is exactly one malicious actor, and given that we have no other information about the participants, then $P(\{P_i \text{ is malicious }\}) = p/N, \forall i$. This is the probability for a member of the DMix group to create a private channel of communication with the malicious actor[9]. This has a good impact on the overall privacy preserving properties.

In fact, assume that the probability $P(I) = 90\%$ and that there are $N = 10$ participants. Therefore the probability of exchanging the address with the malicious actor (and therefore the probability to still be tracked after the use of a private channel and DMix) is 9%. If a participant deems this probability to be too high, he can open multiple private channels with other users and relay other people addresses. For the sake of an example, let's say that P_1 opens three private channels. In this case then, the probability of communicating his own address to the malicious person is

$$\frac{p}{N} \cdot \frac{p}{N-1} \cdot \frac{p}{N-2}$$

which means, following the numeric example above, that P_1 has less than 0.15% of probability of communicating his own address to the malicious person. Furthermore, to

[9]Actually, if this party is sure not to be malicious himself, the probability is slightly higher and it is equal to $k/(N-1)$ for obvious reasons.

be extra cautious, parties can relay addresses using the encryption method used by CoinShuffle [19] which would further increase the uncertainty.

VI. Applications and Conclusion

We proposed DMix, a method to mix coins in a decentralized way that provides unlinkability under exclusive control of the participants. Applications of a decentralized mixer range from gaining privacy for payments (e.g. after business negotiations) to proxy payments[10]. Furthermore, DMix can help regaining the right to be forgotten [32] if decentralized identities will use the blockchains as management support. In fact, DMix is practically able to cut the ties between one transaction and the following one.

Our method resists the currently used heuristics in chain analysis and provide users with better privacy thanks to the common output amount of the `outDMix` transaction. To accomplish this result, we used the MuSig signature aggregation protocol.

References

[1] Ethan Heilman, Leen AlShenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. In *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA, 2017. Internet Society.

[2] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An Empirical Analysis of Anonymity in Zcash. page 16.

[3] Philip Koshy, Diana Koshy, and Patrick McDaniel. An Analysis of Anonymity in Bitcoin Using P2P Network Traffic. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, volume 8437, pages 469–485. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[4] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference - IMC '13*, pages 127–140, Barcelona, Spain, 2013. ACM Press.

[5] Mauro Conti, Sandeep Kumar E, Chhagan Lal, and Sushmita Ruj. A Survey on Security and Privacy Issues of Bitcoin. *IEEE Communications Surveys & Tutorials*, 20(4):3416–3452, 24.

[6] Rui Zhang, Rui Xue, and Ling Liu. Security and Privacy on Blockchain. *arXiv:1903.07602 [cs]*, August 2019.

[7] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world. https://bitcointalk.org/?topic=279249, August 2013.

[8] Muoi Tran, Loi Luu, Min Suk Kang, Iddo Bentov, and Prateek Saxena. Obscuro: A Bitcoin Mixer using Trusted Execution Environments. page 30.

[9] Andrew Poelstra. Mimblewimble. download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

[10] Dr Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. page 32.

[11] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184, San Jose, CA, USA, May 2015. IEEE.

[12] Antoine Rondelet and Michal Zajac. Zeth: On integrating zerocash on ethereum. *arXiv preprint arXiv:1904.00905*, 2019.

[13] DZJ Williamson. The aztec protocol. *URL: https://github.com/AztecProtocol/AZTEC*, 2018.

[10] A proxy payment is done by getting the money from a DMix instead of the payer in order to mask his real address, e.g. for anonymous donations.

[14] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software Grand Exposure: SGX Cache Attacks Are Practical. *arXiv:1702.07521 [cs]*, February 2017.

[15] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves. In Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows, editors, *Computer Security – ESORICS 2016*, volume 9878, pages 440–457. Springer International Publishing, Cham, 2016.

[16] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable Zero Knowledge via Cycles of Elliptic Curves (extended version). page 47.

[17] Pieter Wuille, Jonas Nick, and Tim Ruffing. Schnorr BIP. https://github.com/sipa/bips/blob/bip-schnorr/bip-schnorr.mediawiki.

[18] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, September 2019.

[19] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, volume 8713, pages 345–364. Springer International Publishing, Cham, 2014.

[20] Sarah Meiklejohn and Claudio Orlandi. Privacy-Enhancing Overlays in Bitcoin. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, volume 8976, pages 127–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[21] Tin Tironsakkul, Manuel Maarek, Andrea Eross, and Mike Just. Probing the Mystery of Cryptocurrency Theft: An Investigation into Methods for Taint Analysis. *arXiv:1906.05754 [cs]*, December 2019.

[22] Felix Konstantin Maurer, Till Neudecker, and Martin Florian. Anonymous CoinJoin Transactions with Arbitrary Values. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 522–529, Sydney, Australia, August 2017. IEEE.

[23] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3), 1991.

[24] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.

[25] Gregory Neven, Nigel P Smart, and Bogdan Warinschi. Hash function requirements for schnorr signatures. *Journal of Mathematical Cryptology*, 3(1):69–87, 2009.

[26] Manuel Fersch. *The provable security of elgamal-type signature schemes*. doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2018.

[27] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-Key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security - CCS '06*, pages 390–399, Alexandria, Virginia, USA, 2006. ACM Press.

[28] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. An Analysis of Non-standard Transactions. *Frontiers in Blockchain*, 2:7, August 2019.

[29] Gavin Andresen. Address format for pay-to-script-hash. https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki, 2011.

[30] Peter Todd. OP_CHECKLOCKTIMEVERIFY. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, 2014.

[31] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. page 9.

[32] Christopher Allen. The path to self-sovereign identity. *Life with Alacrity*, 2016.