P2P Mixing and Unlinkable Bitcoin Transactions

Anonymity of the people, by the people, and for the people

Tim Ruffing
CISPA, Saarland University
tim.ruffing@mmci.uni-saarland.de

Pedro Moreno-Sanchez
Purdue University
pmorenos@purdue.edu

Aniket Kate
Purdue University
aniket@purdue.edu

***Abstract*—Starting with Dining Cryptographers networks (DC-net), several peer-to-peer (P2P) anonymous communication protocols have been proposed. Despite their strong anonymity guarantees none of those has been employed in practice so far: Most fail to simultaneously handle the crucial problems of slot collisions and malicious peers, while the remaining ones handle those with a significant increased latency (communication rounds) linear in the number of participating peers in the best case, and *quadratic* in the worst case. We conceptualize these P2P anonymous communication protocols as *P2P mixing*, and present a novel P2P mixing protocol, DiceMix, that only requires constant (i.e., four) communication rounds in the best case, and $4 + 2f$ rounds in the worst case of f malicious peers. As every individual malicious peer can prevent a protocol run from success by omitting his messages, we find DiceMix with its worst-case linear-round complexity to be an optimal P2P mixing solution.**

On the application side, we find DiceMix to be an ideal privacy-enhancing primitive for crypto-currencies such as Bitcoin. The public verifiability of their pseudonymous transactions through publicly available ledgers (or blockchains) makes these systems highly vulnerable to a variety of linkability and deanonymization attacks. DiceMix can allow pseudonymous users to make their transactions unlinkable to each other in a manner *fully compatible* with the existing systems. We demonstrate the efficiency of DiceMix with a proof-of-concept implementation. In our evaluation, DiceMix requires less than 8 seconds to mix 50 messages (160 bits, i.e., Bitcoin addresses), while the best protocol in the literate requires almost 3 minutes in a very similar setting. As a representative example, we use DiceMix to define a protocol for creating unlinkable Bitcoin transactions.

Finally, we discover a generic attack on P2P mixing protocols that exploits the implicit unfairness of a protocol with a dishonest majority to break anonymity. Our attack uses the attacker's real-world ability to omit some communication from a honest peer to deanonymize her input message. We also discuss how this attack is resolved in our application to crypto-currencies by employing uncorrelated input messages across different protocol runs.

I. Introduction

Chaum [15] introduced the concept of anonymous digital communication in the form of mixing networks (or *mixnet*). In the mixnet protocol, a batch of encrypted messages from users are decrypted, randomly permuted, and relayed by a sequence of routers (or proxies) to avoid individual messages from getting traced through the network. The original mixnet protocol as well as its several more successful successor anonymous communication networks (ACNs) such as onion routing [26], AN.ON [2], [5], and Tor [18] inherently require access to a set of geo-politically distributed routers such that at least some of them are trusted to not break users anonymity.

Starting with the dining cryptographers network (DC-net) protocol [14], another line of ACNs research emerged, where users (or peers) do *not* employ any third party proxies and instead communicate with each other to send their messages anonymously. While the DC-net protocol can guarantee successful termination and anonymity against honest-but-curious adversary controlling a subset of users (or peers), it is easily prone to disruption by a single malicious peer who sends invalid protocol messages (active disruption) or omits protocol messages entirely (crash). Moreover, a DC-net protects the anonymity of the involved malicious peers and subsequently cannot ensure termination of the protocol run by detecting and excluding the malicious peer.

To address this termination issue, several recent successors of the DC-net protocol [12], [17], [22], [23], [27], [49] incorporate cryptographic accountability mechanisms against active disruptions. The employed techniques are either preventive, e.g., zero-knowledge proofs proving the validity of sent messages [27], or reactive, e.g, the revelation of session secrets to expose and exclude malicious disruptors after a failed protocol run [17]. Through these measures, these recent successors of DC-net have demonstrated that, for a set of mutually distrusting peers, sending their messages anonymously is *feasible* purely by communicating with each other in a peer-to-peer (P2P) manner, even when a large majority of those are malicious. Moreover, given the surging demand for strong anonymity for P2P cryptocurrency systems such as Bitcoin [1], [3], [34], [39], [40], [44], [46], these schemes have now resulted in some real-world P2P Bitcoin mixing systems [36], [43], [45].

Nevertheless, these feasible solution are still not ideal: with communication rounds linear in the number of participating peers in the best case and quadratic in the worst case, these current pure P2P solutions [17], [45], [49] are not scalable as the number of participating peers grow; e.g., the Bitcoin P2P mixing protocol CoinShuffle [45] requires a few minutes for anonymizing communication of 50 participants. In this paper, we wish to bring the pure P2P anonymous communication protocol from the realm of feasibility to the realm of real-world usage.

A. Contributions

We compartmentalize our contributions in this paper in four key components.

a) P2P Mixing: As our first contribution, we conceptualize peer-to-peer (P2P) mixing as a natural generalization of the dining cryptographers network (DC-net) [14] protocol.

A P2P mixing protocol allows a set of mutually distrusting peers to publish their messages anonymously without requiring any external (trusted or untrusted) party even when all but two of those behave malicious. Motivated from the privacy requirement of real-world P2P payment systems, we design an interface and execution model for P2P mixing protocol that allows easy integration of a P2P mixing protocol instance to the anonymity-seeking payment application.

b) DiceMix Protocol: Although some DC-net successors [25], [35] as well as some anonymous group messaging systems [17], [45], [49] satisfy the P2P mixing requirements, we found those to be not efficient enough for a large-scale mixing. As our second contribution, we present the new P2P mixing protocol DiceMix. DiceMix builds on the original DC-net protocol, and handles collisions and malicious peers using an interesting privacy-preserving redundant messaging step. When every peer behaves honestly, DiceMix requires only a *constant* (i.e., four) number of rounds, and it only increases to $4 + 2f$ rounds in the worst case, even in the presence of f malicious peers. Both of these communication round complexities are a linear factor better than the existing approaches [17], [25], [35], [45].

We also provide a proof-of-concept implementation of the DiceMix protocol, and evaluate it in a global Emulab [52] network scenario. Our results show that even 50 peers can anonymously broadcast their messages in less than 8 seconds, demonstrating that DiceMix is highly practical for cryptocurrency networks where transaction execution takes a few minutes.

c) CoinShuffle++ Protocol: As our third contribution, we instantiate DiceMix with the most prominent P2P payment system: Bitcoin. In particular, building on the CoinJoin paradigm [37] and DiceMix, we present CoinShuffle++, a practical decentralized mixing protocol for the Bitcoin users. CoinShuffle++ not only is considerably simpler and thus easier to implement than CoinShuffle [45] but also inherits the efficiency of DiceMix and thus outperforms CoinShuffle significantly. In particular, in a scenario with 50 participants and a really similar evaluation setting, a successful transaction with CoinShuffle++ can be performed in 8 seconds instead of the almost the 3 minutes required with CoinShuffle. We also find CoinShuffle++ to be compatible with other currently deployed privacy-preserving P2P currency systems and provide detailed description to ease deployment in crypto-currency clients.

Moreover, with its well-defined generic communication interface, DiceMix is applicable to P2P communication systems in general, and we propose it also for non-payment applications such as Sybil-resistant pseudonymization [21].

d) A Generic Attack on P2P Mixing Protocols: As our fourth and final contribution, we provide a generic attack on P2P mixing protocols that exploits the implicit unfairness of a protocol allowing dishonest majority to break the anonymity of a peer. Our attack uses the attacker's real-world ability to omit some communication from a honest peer to deanonymize this contributed message. We exemplify our attack on the Dissent shuffle protocol [17], [49], and also discuss its generality as well as impossibility to avoid it without making any additional assumption similar to the fairness resolution. Our attack draws similar conclusion as those drawn for DoS attacks by Borisov et al. [11] for mixnets and onion routing.

Nevertheless, we also show how this attack can easily be avoided in applications which aim to mix fresh uncorrelated input messages, e.g., coin mixing in the anonymity-seeking crypto-currencies.

B. Organization

The paper is organized as follows. We start by defining the problem of P2P mixing in Section II. We then overview the DiceMix protocol in Section III, while we detail the DiceMix protocol in Section IV and evaluate its performance in Section V. We describe the use of DiceMix to achieve anonymous transactions in Bitcoin in Section VI and discuss related work in Section VII. We describe a deanonymization attack on state-of-the-art P2P mixing protocols in Section VIII and we conclude our work in Section IX.

II. Conceptualizing Peer-to-Peer (P2P) Mixing

In this section, we define the concept of P2P mixing. In particular, we describe the system model, the functionality of a P2P mixing protocol by describing its inputs and outputs, and a detailed execution model. We then describe the threat model, and state the security goals of a P2P mixing protocol.

A. P2P Mixing Protocol

A P2P mixing protocol [17], [45], [55] allows n mutually distrusting peers to simultaneously broadcast their messages in an anonymous manner *without the help of any third-party proxy*, such that an attacker controlling the network and some peers cannot tell which of the fresh messages belongs to which honest peer. In more detail, the anonymity set of an individual honest peer should be the set of all honest participating peers, and we expect the size of this set to be at least two.

The requirement to achieve sender anonymity without the help of any third-party proxy such as an onion router or a mix server makes P2P mixing fundamentally different from most well-known anonymous communication techniques in the literature. Unlike standard techniques such as onion routing or mix cascades, P2P mixing relies on a much weaker trust assumption and is expected to terminate successfully and provide anonymity in the presence of an attacker controlling up to $n - 2$ peers. As a consequence, each peer must be actively involved in the anonymous communication process which comes with inherent restrictions and expense.

B. Setup and Communication Model

We assume that peers are connected via a bulletin board, e.g., a server receiving messages from each peer and broadcasting them to all other peers. We stress that sender anonymity will be required to hold even against a malicious bulletin board colluding with the attacker; the bulletin board is purely a matter of communication.

We assume the bounded synchronous communication setting, where time is divided in fixed communication rounds such that all messages broadcast by a peer in a round are available to the peers by the end of the same round, and absence of a message on the bulletin board indicates that the peer in question failed to send a message during the round.

Such a bulletin board can seamlessly be deployed in practice, and in fact even already deployed Internet Relay Chat (IRC) servers suffice.[1] To be able to tolerate faulty or disruptive bulletin boards, it is possible to add redundancy by running the same instance of the P2P mixing protocol on multiple bulletin boards. The bulletin board can alternatively be implemented by an (early stopping) reliable broadcast protocol [19], [47] if one is willing to accept the increased communication cost.

We assume that all peers willing to participate in a P2P mixing protocol are identified by verification keys of a digital signature scheme, and that the peers know each other's verification keys at the beginning of a protocol execution.

To find other peers willing to mix messages, a suitable bootstrapping mechanism can be used. Note that a malicious bootstrapping mechanism might hinder sender anonymity by preventing honest peers from participating in the protocol and thereby forcing a victim peer to run the P2P mixing protocol with no or only a few honest peers. While this is a realistic threat against any anonymous communication protocol in general, we consider protection against a malicious bootstrapping mechanism orthogonal to our work.

C. Input and Outputs of a P2P Mixing Protocol

Our treatment of a P2P mixing protocol is special with respect to inputs and outputs. Regarding inputs (the messages to mix), allowing the adversary to control up to $n-2$ (i.e., a majority of) peers introduces an unexpected requirement: input messages must be fresh. Regarding outputs, a P2P mixing protocol according to our definitions provides the feature that the peers will have to explicitly agree on the output, i.e., the set of anonymized messages.

1) Freshness of Input Messages: In contrast to state-of-the-art anonymous and terminating P2P mixing protocols such as Dissent [17] and the protocol by Golle and Juels [27], we require that each peer uses a freshly generated bitstring with sufficient entropy, e.g., a freshly generated verification key, as input message to be mixed. Furthermore, if the honest peers exclude a peer from the protocol, e.g., because the peer appears offline, all messages used so far will be discarded. Then, all remaining peers again generate fresh messages and are required to continue the protocol with these messages.

While this seems to be a severe restriction of functionality compared to the aforementioned protocols, a restriction of this kind is in fact necessary to guarantee anonymity. If otherwise peers can arbitrarily choose their messages in a P2P mixing protocol providing termination, the protocol is inherently vulnerable to an attack breaking sender anonymity! We will explain this attack, which works against state-of-the-art P2P mixing protocols and has been overlooked in the literature so far, in detail in Section VIII.

2) Explicit Confirmation of the Output: We observe that anonymity-seeking P2P applications such as coin mixing [37], [45], [55] or identity mixing [21] require that the peers agree explicitly on the outcome of the mixing before it comes into effect, e.g., by collectively signing the set M of anonymized messages.

We call this additional functionality *confirmation* and incorporate it in our model. The form of the confirmation depends on the application and is left to be defined by the application which calls the protocol. For example in coin mixing, the confirmation is a special transaction signed by all peers; we will discuss this in detail in Section VI.

While the protocol cannot force malicious peers to confirm M, those malicious peers should be excluded and the protocol should finally terminate successfully with a proper confirmation by all non-excluded peers.

D. Interface and Execution Model

To deploy a P2P mixing protocol in various anonymity-seeking applications, our generic definition leaves it up to the application to specify exactly how fresh input messages are obtained and how the confirmation on the result is performed. We restrict our discussion here to terminology and a syntactic description of the interface between the anonymity-seeking application and an employed P2P mixing protocol, and leave the semantic requirements to the protocol construction later.

A protocol instance consists of one or several *runs*, each started by calling the user-defined algorithm $\textsc{Gen}()$ to obtain a fresh input message to be mixed. If a run is disrupted, the protocol can exlude peers that turned out to be malicious. Otherwise, the protocol will obtain a candidate result, i.e., a candidate *output set* M of anonymized messages. Then it calls the user-defined *confirmation subprotocol* $\textsc{Confirm}(i, P, M)$, whose task is to obtain confirmation for M from the *final peer set* P of all unexcluded peers. (The first argument i is an identifier of the run.) Possible confirmations range from a signature on M, to a complex task requiring interaction among the peers, e.g., the creation of a treshold signature.

If confirmation can be obtained from everybody, then the run and the P2P mixing protocol *terminates successfully*. Otherwise, $\textsc{Confirm}(i, P, M)$ by convention fails and reports malicious peers deviating from the confirmation steps back to the P2P mixing protocol. In this case, the protocol can start a new run by obtaining a fresh message via $\textsc{Gen}()$; the malicious peers are excluded in this new run.

An example execution is depicted in Fig. 1. Note that while in this example execution, all runs are sequential, this is not a requirement. For improved efficiency, a P2P mixing protocol can perform several runs concurrently, e.g., to have an already started second run in reserve in case the first fails. Then the protocol can terminate with the first run that confirms successfully, and abort all other runs.

E. Threat Model

In general, we assume that the attacker controls a number f of the peers.

For the sender anonymity property, we assume that the attacker additionally controls the bulletin board, i.e., the network. In particular, the attacker can partition the network and block some messages from a honest peers. In case of successful termination, the anonymity set of each honest peer

[1] Servers supporting IRC version 3.2 are capable of adding a server timestamp to every message [48]; this can ensure that peers agree whether a certain message arrived in time.

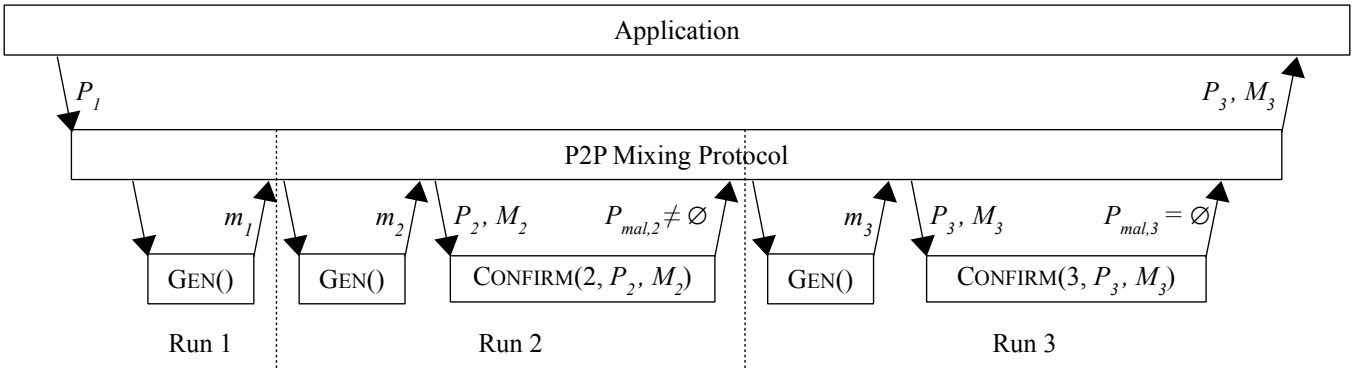

Fig. 1: Example Execution of a P2P Mixing Protocol. The figure shows the calls during the execution; time proceeds from left to right. The execution starts with the application calling the P2P mixing protocol with an initial set P_1 of peers. The P2P mixing protocol then starts Run 1 by generating a new message m_1 (via calling GEN()). Run 1 fails early (e.g., due to active disruption by a peer p) and m_1 is discarded. The P2P mixing protocol then starts Run 2 with peer set $P_2 = P_1 \setminus \{p\}$ by generating a new message m_2. Run 2 is initially not disrupted, and the P2P mixing protocol calls the confirmation subprotocol to confirm the set M_2 of mixed messages with the peers in P_2. The confirmation subprotocol fails, because a set $P_{mal,2}$ of peers refuse to confirm. The confirmation subprotocol reports those malicious peers back to the P2P mixing protocol, which in turn discards m_2. The P2P mixing protocol then starts Run 3 with peer set $P_3 = P_2 \setminus P_{mal,2}$. This time, the confirmation subprotocol succeeds and indicates that by returning an empty set (of malicious peers) to the P2P mixing protocol. That is, all peers in P_3 have confirmed that the set M_3 of anonymized messages is the final output. The P2P mixing protocol returns P_3 and M_3 to the application and terminates.

will be the set of unexcluded honest peers[2]. This means that we need $f < n - 1$ at the end of the protocol, where n is the number of unexcluded peers, to ensure that at least two honest peers are present and the anonymity guarantee is meaningful.

For the termination property, will trust the bulletin board to relay messages reliably and honestly, because termination (or any liveness property) is naturally impossible to achieve against a malicious bulletin board, which can just block all communication.

F. Security Goals

We are now ready to state the security properties of a P2P mixing protocol.

Sender Anonymity: If the protocol succeeds for honest peer p in a run (as described in Section II-D) with message m_p and final peer set P, and $p' \in P$ is another honest peer, then the attacker cannot distinguish whether message m_p belongs to p or to p'.

Termination: If the bulletin board is honest, the protocol eventually terminates successfully for every honest peer.

Note that our definition of sender anonymity is only concerned with the messages in a successful run, i.e., no anonymity is guaranteed for messages discarded in failed runs (see Section II-D). This demands explanation, because giving up anonymity in case of failed confirmation seems to put privacy at risk at first glance. However, the discarded messages are just randomly generated bitstings and have never been and will never be used outside the P2P mixing protocol; in particular no confirmation took place. So it is safe to give up sender anonymity for discarded messages. It turns out that this permissive definition is sufficient for a variety of applications and allows for very efficient constructions.

[2]A honest peer might appear offline due to the attacker blocking network messages. Such a peer can be excluded to allow the remaining peers to proceed.

III. Solution Overview

Our core tool to design an efficient P2P mixing protocol is a Dining Cryptographers Network (DC-net) [14]. We describe an example of a DC-net involving two users. Suppose that two peers p_1 and p_2 share a key k and that one of the peers (e.g., p_1) wishes to anonymously publish a message m such that $|m| = |k|$. For that, p_1 publishes $M_1 := m \oplus k$ and p_2 publishes $M_2 := k$. An observer can compute $M_1 \oplus M_2$, effectively recovering m. The origin of this message is hidden: without knowing the secret k, the observer cannot determine which peer published m. We refer the reader to [27] for details on how to extend this basic protocol to multiple users.

Besides the need for pairwise symmetric keys, which can be overcome by a key exchange mechanism, there are two key issues to deploy a DC-net in practice with an arbitrary number of peers, namely first making it possible that all peers can publish a message simultaneously, and second, ensuring termination of the protocol even in the presence of malicious disruptors, while preserving anonymity.

A. Handling Collisions

Each peer $p \in P$ in the mixing seeks to anonymously publish her own message m_p. Naively, they could run $|P|$ instances of a DC-net, where each peer randomly selects one instance (or slot) to publish her message. However, even if all peers are honest, two peers can choose the same slot with high probability, and their messages are unrecoverable [27].

One proposed solution consists on performing an anonymous slot reservation mechanism so that peers agree in advance on an ordering for publishing [25], [35]. However, this mechanism adds communication rounds among the peers and these reservation schemes must handle collisions on themselves. Alternatively, it is possible to set many more slots so that probability of collision decreases [16]. However, this becomes

inefficient quickly, and two honest peers could still collide with some probability.

Instead, we follow the paradigm of handling collisions by redundancy [12], [20]. Assume that messages to be mixed are encoded as elements of a finite field $\mathbb{F}$ with characteristic greater than the number n of peers. Given n slots, each peer i, with message m_i, publishes m_i^j (i.e., m_i to the j-th power) in the j-th slot. This results in having a collision from all peers for each of the slots. Using addition in $\mathbb{F}$ instead of XOR to create DC-net messages, the j-th collision results on the i-th power sum $S_i = \sum_i m_i^j$.

Now, we require a mechanism to extract the messages m_i from the power sums S_i. Let assume that there are n peers in a run of the DiceMix protocol. At the end of it, the peers can compute the power sums:

$$\begin{aligned} P_1 &= x_1 + x_2 + \ldots + x_n \\ P_2 &= x_1^2 + x_2^2 + \ldots + x_n^2 \\ &\ldots \\ P_n &= x_1^n + x_2^n + \ldots + x_n^n \end{aligned}$$

This can be then used to obtain the values of $x_1, \ldots x_n$ using Newton's identities [28]. For that, let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ be a polynomial such that $f(x) = 0$ have roots $x_1, x_2, \ldots, x_n$. Now, we know that $a_n = 1$ given that it is the coefficient of x^n resulting from the product of $(x - x_1) \cdot \ldots \cdot (x - x_n)$. Newton's identities state that

$$\begin{aligned} P_1 + a_{n-1} &= 0 \\ P_2 + a_{n-1}P_1 + 2a_{n-2} &= 0 \\ P_3 + a_{n-1}P_2 + a_{n-2}P_1 + 3a_{n-3} &= 0 \\ &\ldots \end{aligned}$$

From here, we know all the P_i so that we can easily recover the a_i. Now, once we know the coefficients for the polynomial $f(x)$, we can factorize it. The n roots are the n values that we were looking for.

B. Handling Disruption and Ensuring Termination

Recovering the messages only works when all peers honestly follow the protocol. However, a peer could disrupt the DC-net by simply using inconsistent DC-net messages. Then we must ensure that the protocol will still eventually terminate successfully.

When a candidate set M is determined, every honest peer checks whether her input message is indeed in M. Depending on the outcome of this check, the peer either starts the confirmation subprotocol to confirm a good M, or reveals the secret key used in the key exchange to determine who is responsible for an incorrect M. We face two challenges on the way to successful termination.

1) Consistent Detection of Disruption: The first challenge is to ensure that indeed M does not contain any honest message. Only then all honest peers will agree on whether disruption has occurred and take the same control flow decision at this stage of the protocol, which is crucial for termination

To overcome this challenge, every peer must provide a non-malleable commitment (e.g., using a hash function) to its DC-net vector before it sees the vectors of other peers. In this manner, malicious peers are forced to create their DC-net vectors independently of the DC-net vector (and the unpredictable input messages) of honest peers. Thus, the redundant encoding of messages ensures that a malicious peer is not able to create a malformed DC-net vector that results in a distortion of only a subset of the honest peers. Intuitively, to distort some messages but keep some other message m of a honest peer intact, the malicious peer must influence all power sums consistently. This, however, would require a DC-net vector that depends on m (as we show in Section IV-D), which is prevented by the non-malleability of the commitments. This ensures that all honest peers agree on whether M is correct or not, and take the same control flow decision.

2) Exposing a Disruptor: The second challenge is that the misbehaving peer is not trivially detected given the sender anonymity property of DC-nets. To overcome this, every peer is required to reveal the secret key used in the initial key exchange. Then every peer can replay the steps done by every other peer and eventually detect and expel the misbehaving peer from a new run.

Note that the revelation of the secret keys clearly breaks sender anonymity for the current run of the protocol. However, the failed run will be discarded and a new run with fresh cryptographic keys and fresh messages will be started without the misbehaving peer. This is in line with our definition of sender anonymity, which does not impose a requirement on failed runs.

An important guarantee provided by DiceMix is that if a protocol run fails, the honest peers agree on the set of malicious peers to be excluded. Although this is critical for termination, this aspect has not been properly formalized or addressed in some previous P2P mixing protocols [17], [45], [49] supposed to ensure termination.

IV. The DiceMix protocol

In this section we present DiceMix, an efficient P2P mixing protocol, which terminates in only $4+2f$ rounds in the presence of f malicious peers.

A. Building Blocks

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

2) Non-interactive Key Exchange: We require a non-interactive key exchange (NIKE) mechanism ($\mathsf{NIKE.KeyGen}$, $\mathsf{NIKE.SharedKey}$) secure in the CKS model [13], [24]. The algorithm $\mathsf{NIKE.KeyGen}(id)$ outputs a public key npk and a secret key nsk for given a party identifier id. $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid)$ outputs a shared key for the two parties and session identifier sid. $\mathsf{NIKE.SharedKey}$ must fulfill the standard correctness requirement that for all session identifiers sid, all parties id_1, id_2, and

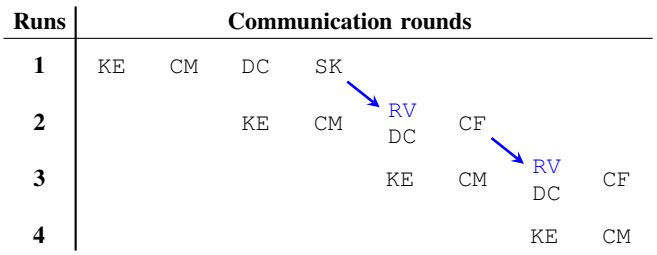

Fig. 2: Example for a DiceMix execution. Run 1 fails due to DC-net disruption. Run 2 fails to confirm. Run 3 finally succeeds and run 4 is then aborted. Rows represent protocol runs and columns represent communication rounds. Blue parts are for concurrency; the arrows depict the dependency between runs, i.e., some run informs the next run about the peers to exclude. KE: Key exchange; CM: Commitment; DC: DC-net; RV: Reveal pads; SK: Reveal secret key; CF: Confirmation.

all corresponding key pairs (npk_1, nsk_1) and (npk_2, nsk_2), it holds that $\mathsf{NIKE.SharedKey}(id_1, id_2, nsk_1, npk_2, sid) = \mathsf{NIKE.SharedKey}(id_2, id_1, nsk_2, npk_1, sid)$. Additionally, we require an algorithm $\mathsf{NIKE.ValidatePK}(npk)$, which outputs *true* if and only if npk is a public key in the output space of $\mathsf{NIKE.KeyGen}$, and we require an algorithm $\mathsf{NIKE.ValidateKeys}(npk, nsk)$ which outputs *true* iff nsk is a secret key for the public key npk.

Static Diffie-Hellman key exchange satifies these requirements [13], given a suitable key derivation algorithm such as $\mathsf{NIKE.SharedKey}(id_1, id_2, x, g^y) := \mathsf{K}((g^{xy}, \{id_1, id_2\}, sid))$ for a hash function K modeled as a random oracle.

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

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

Message x is broadcast using "**broadcast** x". The command "**receive** $\mathrm{X}[p]$ **from all** $p \in P$ **where** $X(\mathrm{X}[p])$ **missing** $C(P_{off})$" attempts to receive a message from all peers $p \in P$. The first message $\mathrm{X}[p]$ from peer p that fulfills predicate $X(\mathrm{X}[p])$ is accepted and stored as $\mathrm{X}[p]$; 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 peers that did not send a (valid) message.

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

B. Contract with the Application

In the following, we specify the contract between the DiceMix and the application calling it. We start with two guarantees provided by DiceMix to the application and then we describe features required on the application by DiceMix.

1) Guarantees Provided to the Application: The confirmation subprotocol is provided with two guarantees. First, DiceMix ensures that all honest peers call the confirmation subprotocol in the same communication round with the same parameters; we call this property *agreement*.

Second, to ensure that no peer can refuse confirmation for a legitimate reason, e.g., an incorrect set M not containing her message, our protocol ensures that all honest peers deliver the same and correct message set M. Consequently, the confirmation subprotocol $\textsc{Confirm}(i, P, M)$ can safely assume that peers refusing to confirm are malicious. We call this *validity*.

The purpose of both of these guarantees is to ensure correct functionality of the confirmation subprotocool, and the guarantees are only provided if the bulletin board is honest. As a consequence, it is up to the confirmation subprotocol to fail safely if they do not hold.

a) Agreement: Assume that the bulletin board is honest. Let p and p' be two honest peers in a protocol instance. If p calls $\textsc{Confirm}(i, P, M)$[3] in some communication round r, then p' calls $\textsc{Confirm}(i, P, M)$ with the same message set M and final peer set P in the same communication round r.

b) Validity: Assume that the bulletin board is honest. If honest peer p calls $\textsc{Confirm}(i, P, M)$ with message set M and final peer set P, then *(i)* for all honest peers p' and their messages $m_{p'}$, we have $m_{p'} \in M$, and *(ii)* we have $|M| = |P|$.

2) Requirements on the Application: Next, we specify the guarantees that the application must provide to DiceMix to ensure proper function.

We assume that input messages generated by $\textsc{Gen}()$ are encoded in a prime field $\mathbb{F}_q$, where q is larger than the number of peers in the protocol. Also, we assume w.l.o.g. that the message m returned by $\textsc{Gen}()$ has sufficient entropy such that it can be predicated only with negligible probability. (This can be ensured by a randomized encoding such as encoding m as $m||r$ for a sufficiently large string r.) Note that this in particular implies that q is at least as large as the security parameter.

We require two natural properties from the confirmation subprotocol. The first property (*correct confirmation*) states that a successful call to the subprotocol indeed confirms that the peers in P agree on M. The second property (*correct exclusion*) states that in an unsuccessful call, the confirmation subprotocol identifies at least one malicious peer, and no honest peer is falsely identified as a malicious peer.

a) Correct Confirmation: Even if the bulletin board is malicious,[4] we require the following: If a call to $\textsc{Confirm}(i, P, M)$ succeeds for peer p (i.e., if the call returns an empty set $P_{malicious} = \emptyset$ of malicious peers refusing confirmation), then all honest peers in P have called $\textsc{Confirm}(i, P, M)$.

[3] $\textsc{Confirm}(\ldots)$ will actually take more arguments but they are not relevant for this subsection.

[4] This property puts forth a requirement on a successful call of the confirmation subprotocol. Such a successful call will result in a successful run and ultimately in a successful termination of the whole P2P mixing protocol, which implies that the messages are not discarded and sender anonymity is required for this run. So this property is crucial for sender anonymity and thus we must assume that it holds even if the bulletin board is malicious.

b) Correct Exclusion: Assume that the bulletin is honest. If CONFIRM(i, P, M) returns a set $P_{malicious} \neq \emptyset$ for peer p, then CONFIRM(i, P, M) returns the same set $P_{malicious}$ for every honest peer p'. Furthermore, the returned set $P_{malicious}$ does not contain honest peers.

C. Protocol Description

We describe the DiceMix protocol in Algorithms 1 and 2.

The black code is the basic part of the protocol; the blue code handle several concurrent runs offline peers.

1) Single Run of the Protocol (Black Pseudocode): The protocol starts in START-DICEMIX(), which receives as input a set of other peers P, our own identity *my*, an array VK[] of verification keys of the other peers, our own signing key sk, and a predetermined session identifier sid.

A single run of DiceMix consists of four rounds. (RUN()) The first three rounds are key exchange (KE), publishing of commitments to the DC-net messages (CM) and publishing of such DC-net messages (DC). The fourth round consists on accepting the result in the absence of disruptions (CF) or the publishing of secret keys to discover the misbehaving peer (SK).

The first round (KE) just uses the NIKE to establish pairwise symmetric keys between all peers (DC-KEYS()). Then each peer can derive the DC-net pads from these symmetric keys (DC-PAD()).

In the second round (CM), each peer commits to her DC-net vector using hash function H; adding randomness is not necessary, because we assume that the input messages contained in the DC-net vector have sufficient entropy. The commitments are opened in the third round (DC). They are non-malleable and their purpose is to prevent a rushing attacker from letting his DC-net vector depend on messages by honest peers, which will be crucial for the agreement property. After opening the commitments, every peer has enough information to solve the DC-net and extract the list of messages by solving the power sums (DC-RES()).

Finally, every peer checks whether her input message is in the result of the DC-net. Agreement will ensure that either every peer finds her message or no honest peer finds it.

If a peer finds her message, she proceeds to the confirmation subprotocol. Otherwise, she outputs her secret key. In this case, every other peer publishes her secret key as well, and all peers can replay each other protocol messages for the current run. This will expose the misbehaving peer and honest peers will exclude him from the next run.

2) Concurrent Runs of the Protocol (Blue Pseudocode): A simple but inefficient way of having several runs is to start a single run of the protocol and only after misbehavior is detected, start a new run without the misbehaving peer. This approach requires $4 + 4f$ rounds, where f is the number of disruptive peers (assuming that CONFIRM() takes one round). To reduce the number of communication rounds to $4 + 2f$, we deploy concurrent runs as exemplified in Fig. 2. We need to address two main challenges. First, when a peer disrupts the DC-net phase of run i, it must be possible to "patch" the already started run $i+1$ to discard messages from misbehaving peers in run i. For that, run i must reach the last phase (SK or CF) before run $i + 1$ reaches DC phase.

Until run $i+1$ sends the DC message, it can can be patched as follows. In the DC phase of run $i+1$, honest peers broadcast not only their DC-net messages, but also in parallel they reveal (RV) the symmetric keys shared in run $i + 1$ with malicious peers detected in run i. In this manner, DC-net messages can be partially unpadded, effectively excluding misbehaving peers from run $i + 1$. We note that, a peer could reveal wrong symmetric keys in the RV step. This, however, leads to wrong output of the DC-net, which is then handled by revealing secret keys in round $i + 1$. Moreover, publishing partial symmetric keys does not compromise sender anonymity since messages remain partially padded with symmetric keys shared between the honest peers.

3) Handling Offline Peers (Blue Pseudocode): So far we have only discussed how to ensure termination against actively disruptive peers who send wrong messages. However, a malicious peer can also just send no message at all. This case is easy too handle in our protocol. If a peer p has not provided a (valid) broadcast message to the bulletin board (in time), all honest peers will agree on that fact, and exclude the unresponsive peer. In particular, it is easy to see that all criteria specifying whether a message is valid will evaluate the same for all honest peers (if the bulletin board is reliable, which we assume for termination).

Observe that for missing messages from the first two broadcasts (KE and CM), the current run can be continued. Peers not sending KE are just ignored in the rest of the run; peers not sending CM are handled by revealing symmetric keys exactly as done with concurrent runs. Observe that this crucially ensures that even in the presence of passively disrupting peers, only $4 + 2f$ communications rounds are necessary.

D. Security and Correctness Analysis

In this section, we discuss why DiceMix achieves all required properties, namely the security properties sender anonymity and termination as well as the guarantees validity and agreement that the application may rely on.

1) Sender Anonymity: Consider a protocol execution in which a honest peer p succeeds with message m_p and final peer set P, and let $p' \in P$ be another honest peer. We have to argue that the attacker cannot distinguish whether m_p belongs to p or p'.

Since both p and p' choose fresh messages m_p, $m_{p'}$, and fresh NIKE key pairs in each run, it suffices to consider only the successful run i. Since p succeeds in run i, the call to CONFIRM(i, P, M) has succeeded. By the "correct confirmation" property of CONFIRM$(\ldots)$, p' has started CONFIRM(i, P, M) in the same communication round as p. By construction of the protocol, this implies two properties about p': *(i)* p' will not reveal her secret key in round SK. *(ii)* peer p' assumes that p is not excluded in run i, and thus has not revealed the symmetric key shared with p in round RV. (Part *(ii)* is only relevant in the concurrent variant of the protocol.)

As the key exchange scheme is secure in the CKS model and the public keys are authenticated using signatures, the

Algorithm 1 DiceMix: Main Protocol Run

procedure START-DICEMIX($P, my, \text{VK}[\,], sk, sid$)
  $sid' := (sid, P, \text{VK}[\,])$
  if $my \in P$ **then**
    fail "cannot run protocol with myself"
  RUN($P, my, \text{VK}[\,], sk, sid', 0$)

procedure RUN($P, my, \text{VK}[\,], sk, sid, run$)
  if $P = \emptyset$ **then**
    fail "no honest peers"
  ▷ Exchange pairwise keys
  $(\text{NPK}[my], \text{NSK}[my]) := \mathsf{NIKE.KeyGen}(my)$
  $sidH := \mathsf{H}((\texttt{sid}, sid, P \cup \{my\}, \text{NPK}[\,], run))$
  broadcast $(\texttt{KE}, \text{NPK}[my], \mathsf{Sign}(sk, (\text{NPK}[my], sidH)))$
  receive $(\texttt{KE}, \text{NPK}[p], \sigma[p])$ **from all** $p \in P$
    where $\mathsf{NIKE.ValidatePK}(\text{NPK}[p])$
      $\wedge\ \mathsf{Verify}(\text{VK}[p], \sigma[p], (\text{NPK}[p], sidH))$
  missing P_{off} **do**
    $P := P \setminus P_{off}$ ▷ Exclude offline peers
  ▷ Create fresh message to mix and prepare DC-net
  $m := \text{GEN}()$
  $\text{K}[\,] := \text{DC-KEYS}(P, \text{NPK}[\,], my, \text{NSK}[my], sidH)$
  $\text{DC}[my][\,] := \text{DC-VECTOR}(P, \text{K}[\,], m)$
  $P_{excl} := \emptyset$ ▷ Malicious (or offline) peers for later exclusion
  ▷ Commit to DC-net vector
  $\text{COM}[my] := \mathsf{H}((\texttt{dc}, \text{DC}[my]))$
  broadcast $(\texttt{CM}, \text{COM}[my], \mathsf{Sign}(sk, (\text{COM}[my], sidH)))$
  receive $(\texttt{CM}, \text{COM}[p], \sigma[p])$ **from all** $p \in P$
    where $\mathsf{Verify}(\text{VK}[p], \sigma[p], (\text{COM}[p], sidH))$
  missing P_{off} **do** ▷ Store offline peers for exclusion
    $P_{excl} := P_{excl} \cup P_{off}$
  if $run > 0$ **then**
    ▷ Wait for prev. run to notify us of malicious peers
    $P_{exclPrev} :=$ **wait**
    $P_{excl} := P_{excl} \cup P_{exclPrev}$
  ▷ Collect shared keys with excluded peers
  for all $p \in P_{excl}$ **do**
    $\text{K}_{excl}[my][p] := \text{K}[p]$
  ▷ Start next run (in case this one fails)
  $P := P \setminus P_{excl}$
  $next :=$ **fork** RUN($P, my, \text{VK}[\,], sk, sid', run + 1$)
  ▷ Open commitments and keys with excluded peers
  broadcast $(\text{DC}, \text{DC}[my][\,], \text{K}_{excl}[my][\,], \mathsf{Sign}(sk, \text{K}_{excl}[my][\,]))$
  receive $(\text{DC}, \text{DC}[p][\,], \text{K}_{excl}[p][\,], \sigma[p])$ **from all** $p \in P$
    where $\mathsf{H}((\texttt{dc}, \text{DC}[p][\,])) = \text{COM}[p]$
      $\wedge\ \{p' : \text{K}_{excl}[p][p'] \neq \bot\} = P_{excl}[p]$
      $\wedge\ \mathsf{Verify}(\text{VK}[p], \text{K}_{excl}[p][\,], \sigma[p])$
  missing P_{off} **do** ▷ Abort and rely on next run
    return RESULT-OF-NEXT-RUN($P_{off}, next$)
  ▷ Check if our output is contained in the result
  $M := \text{DC-RES}(P \cup \{my\}, \text{DC}[\,][\,], P_{excl}, \text{K}_{excl}[\,][\,])$
  if $m \in M$ **then**
    $P_{mal} := \text{CONFIRM}(i, P, M, my, \text{VK}[\,], sk, sid)$
    if $P_{mal} = \emptyset$ **then** ▷ Success?
      abort *next*
      return m
  else
    broadcast $(\texttt{SK}, \text{NSK}[my])$ ▷ Reveal secret key
    receive $(\texttt{SK}, \text{NSK}[p])$ **from all** $p \in P$
      where $\mathsf{NIKE.ValidateKeys}(\text{NPK}[p], \text{NSK}[p])$
    missing P_{off} **do** ▷ Abort and rely on next run
      return RESULT-OF-NEXT-RUN($P_{off}, next$)
    ▷ Determine malicious peers using the secret keys
    $P_{mal} := \text{BLAME}(\text{NPK}[\,], \text{NSK}[\,], \text{DC}[\,][\,], sidH, P_{excl}, \text{K}_{excl}[\,][\,])$
  return RESULT-OF-NEXT-RUN($P_{mal}, next$)

Algorithm 2 DiceMix: Sub-procedures

procedure DC-VECTOR($P, \text{K}[\,], m$)
  ▷ DC-net
  for $s := 1, \ldots, |P| + 1$ **do**
    $\text{DCMY}[s] := (m)^s + \text{DC-PAD}(P, \text{K}[\,], s)$
  return $\text{DCMY}[\,]$

procedure DC-KEYS($P, \text{NPK}[\,], my, nsk, sidH$)
  for all $p \in P$ **do**
    $\text{K}[p] := \mathsf{NIKE.SharedKey}(my, p, nsk, \text{NPK}[p], sidH)$
  return $\text{K}[\,]$

procedure DC-PAD($P, \text{K}[\,], s$)
  return $\sum_{p \in P} \text{sgn}(my - p) \cdot \mathsf{G}((\text{K}[p], s))$ ▷ in $\mathbb{F}$

procedure DC-RES($P_{all}, \text{DC}[\,][\,], P_{excl}, \text{K}_{excl}[\,][\,]$)
  for $s := 1, \ldots, |P| + 1$ **do**
    ▷ Pads cancel out for honest peers
    $\text{S}[s] := \sum_{p \in P_{all}} \text{DC}[p][s]$
    ▷ Also remove pads for excluded peers
    $\text{S}[s] := \text{S}[s] - \sum_{p \in P_{all}} \text{DC-PAD}(P_{excl}, \text{K}_{excl}[p], s)$
  $\text{M}[\,] := \mathsf{Solve}(\forall s \in \{1, \ldots, |P| + 1\}.\ \text{S}[s] = \sum_{i=1}^{|P|+1} \text{M}[i]^s)$
  return $\mathsf{Set}(\text{M}[\,])$ ▷ Convert $\text{M}[\,]$ to an (unordered) set

procedure BLAME($\text{NPK}[\,], \text{NSK}[\,], \text{DC}[\,][\,], sidH, P_{excl}, \text{K}_{excl}[\,][\,]$)
  $P_{mal} := \emptyset$
  for all $p \in P$ **do**
    $\text{K}'[\,] := \text{DC-KEYS}(P, \text{NPK}[\,], p, \text{NSK}[p], sidH)$
    ▷ Purported m of p
    $m' := \text{DC}[p][1] - \text{DC-PADS}(\text{K}[\,], 1)$
    ▷ Replay DC-net message of p
    $\text{DC}'[\,] := \text{DC-VECTOR}(\text{K}'[\,], m')$
    if $\text{DC}'[\,] \neq \text{DC}[p][\,]$ **then**
      $P_{mal} := P_{mal} \cup \{p\}$ ▷ Exclude inconsistent p
    ▷ Check if p published correct symmetric keys
    for all $p_{excl} \in P_{excl}$ **do**
      if $\text{K}_{excl}[p][p_{excl}] \neq \text{K}'[p_{excl}]$ **then**
        $P_{mal} := P_{mal} \cup \{p\}$
  return P_{mal}

procedure RESULT-OF-NEXT-RUN($P_{exclNext}, next$)
  ▷ Hand over to next run and notify of peers to exclude
  notify *next* **of** $P_{exclNext}$
  ▷ Return result of next run
  result := **join** *next*
  return *result*

attacker cannot distinguish the random DC-nets derived from the symmetric key between p and p' from random pads.

Thus, after opening the commitments on the pads, p has formed a proper DC-net with at least p'. The security guarantee of original Chaum's DC-nets [14] implies that the attacker cannot distinguish m_p from $m_{p'}$ before the call to CONFIRM(i, P, M). Now, observe that the execution of subprotocol CONFIRM(i, P, M) does not help in distinguishing, since all honest peers call it with the same arguments, which follows by the "correct confirmation" property as we have already argued. This shows sender anonymity.

2) Validity: To show validity, we have to show that if honest peer p calls CONFIRM(i, P, M) with message set M and final peer set P, then *(i)* for all honest peers p' and their messages $m_{p'}$, we have $m_{p'} \in M$, and *(ii)* we have $|M| = |P|$.

For the first part of validity, recall that we assume the bulletin board to be honest for validity, so every peer receives

the same broadcast messages. Under this assumption and the assumption that the signature scheme is unforgeable, a code inspection shows that after receiving the DC message, the entire state of a protocol run i is the same for every honest peer, except for the signing keys, the own identity *my*, and the message m generated by GEN(). From these three items, only m influences the further state and control flow, and it does so only in the check $m \in M$ at the end of RUN(...).

We now show as intermediate step that in every run i, the condition $m \in M$ (in the last part of RUN(...)) is either true for all honest peers or is false for all honest peers. Note that also M is entirely determined by broadcast messages and thus the same for all honest peers. Let p and p' be two honest peers with their input messages m_p and $m_{p'}$ in run i, and assume for contradiction that the condition is true for p but not for p', i.e., $m_p \in M$ but $m_{p'} \notin M$. This implies that at least one malicious peer a has committed to an ill-formed DC-net vector in run i, i.e., a vector which is not of the form $(m_a, m_a^2, \ldots, m_a^n)$ with $n \geq 2$. Since $m_p \in M$, this ill-formed vector left the message m_p intact. This implies that the attacker has information about the other DC-net vectors. A simple algebraic argument shows that even for the second power, it is not feasible to come up with additive offset to the power sum that changes some of the encoded messages but leaves others intact; to change some $m_{p'}$ to $m'_p + \delta$, knowledge of $(m_{p'} + \Delta)^2 - m_{p'}^2 = 2m_{p'}\Delta + \Delta^2$ and thus knowledge of $m_{p'}$ is necessary.

As the message $\mathsf{H}(\texttt{dc}, \text{DC}[\,])$ implements a hiding, binding and non-malleable commitment on DC[] (recall that adding randomness is not necessary because there is sufficient entropy in DC[]), it is infeasible, even for a rushing malicious peer a, to have committed to an ill-formed vector that leaves m_p intact. This is a contradiction, and thus the condition $m \in M$ evaluates equivalently for all honest peers.

Now observe that the condition $m \in M$ determines whether CONFIRM(...) is called. That is, whenever CONFIRM(i, P, M) is called by some honest peer p, then $m_{p'} \in M$ for all honest peers p'. This shows the first part of validity.

For the second part of validity ($|M| = |P|$), observe that in the beginning of an execution and whenever P changes, a new run with $|P|$ peers is started, each of which submits exactly one message. This shows validity.

3) Agreement: To show agreement, we have to show that for all runs i, if one honest peer p calls CONFIRM(i, P, M) in some round, then all honest peers p' call CONFIRM(i, P, M) in the same round. This follows from validity. By the first part of validity, we know that some honest peer calls CONFIRM(i, P, M), then $m_{p'} \in M$ for all honest peers p' in run i. By construction of the protocol, $m_{p'} \in M$ this is exactly the condition that determines whether p' calls CONFIRM(i, P, M). Thus all honest peers p' call CONFIRM(i, P, M) in the same round.

4) Termination: Now, we show why the protocol terminates for every honest peer. We first show that at least one malicious peer is excluded in each failed run. We have already argued above (for validity) that in the presence of an honest bulletin board, all honest peers take the same control flow decision (whether to call CONFIRM(...) or not at the end of each run). We can thus distinguish cases on this control flow decision.

If CONFIRM(...) is called in a failed run, then it returns the same non-empty set of malicious peers (by the "correct exclusion" property), and those peers will be excluded by every honest peer. If CONFIRM(...) is not called in a run, then there must have been disruption by at least one malicious peer. Replaying all protocol messages of this run (with the help of then published secret keys) clearly identifies at least one malicious peer; and since all honest peers run the same code (BLAME(...)) on the same inputs to do so, they will all exclude the same set of malicious users.

We have shown that in each failed run, all honest peers exclude the same non-empty set of malicious peers. Eventually, we will reach one of two cases. In the first case, the number of unexcluded peers will drop below 2; in that case the protocol is allowed to fail and thus there is nothing to show. In the second case, we will reach a run in which all peers behave honestly (whether they are controlled by the attacker or not). This run will successfully terminate, which shows termination.

E. Variants of the Protocol

The design of DiceMix follows the P2P paradigm, and consequently, we do not expect the bulletin board to implement any real functionality or perform any computation. The bulletin board is a simple broadcast mechanism and may be replaced by a suitable reliable broadcast protocol [47].

However, if one is willing to require a more sophisticated bulletin board with dedicated functionality, the efficiency of DiceMix can be improved. (In that case, calling it a server may be more appropriate; we stick with bulletin board for simplicity.) It is important to note that even a dedicated bulletin board is still only trusted for termination and not for anonymity.

1) Dropping the Commitment Phase: Recall that the purpose of the non-malleable commitments is to prevent malicious peers from choosing their DC vectors depending on the DC vectors of the honest peers.

Assume that the bulletin board supports secure channels, and broadcasts the messages in the DC round only after all peers have submitted their messages. Then independence is ensured with a honest bulletin board, and we can drop the CM (commitment) round. This is secure because the independence of the DC vectors is only necessary for termination but not for anonymity, and we trust the bulletin board for termination already. A serial protocol execution (without concurrency) will then follow the pattern "KE (DC CF/SK)+", where the plus indicates that these phases are performed once or several times. With the help of concurrency, we can run the key exchange (KE) concurrently to the confirmation phase (CF/SK), and reduce the number of rounds to $3 + 2f$ (assuming that the confirmation phase takes one round.). An example run is depicted in Fig. 3.

Note that a revelation of symmetric keys (RV in the original protocol) will not be necessary anymore, because the malicious peers to exclude are determined before the DC round of second run (see Section IV-C2 for an explanation of RV).

a) Bulletin Board Performs Expensive Computation: Moreover, a dedicated bulletin board can perform the expensive computation of solving the equation system involving the power sums, and broadcast the result instead of the DC vectors. The

Runs	Communication rounds								
1	KE	DC	SK						
2			KE	DC	CF				
3					KE	DC	CF		
4							KE		

Fig. 3: Example for a DiceMix execution with a dedicated bulletin board. Run 1 fails due to DC-net disruption. Run 2 fails to confirm. Run 3 finally succeeds and run 4 is then aborted. Rows represent protocol runs and columns represent communication rounds. The blue arrows depict dependencies between runs, i.e., some run informs the next run about the peers to exclude. `KE`: Key exchange; `CM`: Commitment; `DC`: DC-net; `SK`: Reveal secret key; `CF`: Confirmation.

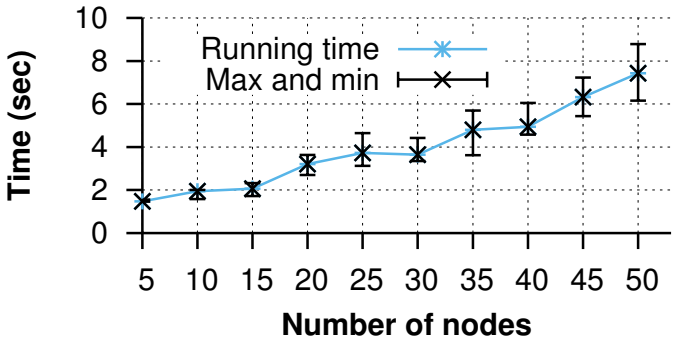

Fig. 4: Running Time (average and minimum/maximum over all peers). All peers have a bandwith of 10 Mbit/s; the bulletin board has a total of 1 Gbit/s; all links have 50 ms latency.

bulletin board would then also be responsible for handling inconsistent messages in the `SK` run; it would then announce the malicious peers after having received all secret keys. This saves communication in the rounds `DC` and `SK`. Again, security is preserved, because we trust bulletin board for termination.

V. Performance Analysis

In this section, we analyze the performance of DiceMix. We first analyze the communication costs, and then evaluate the running time with the help of a prototype implementation. Our results show that DiceMix is practical and outperforms existing solutions.

Using concurrent runs, DiceMix needs $(c+3)+(c+1)f$ communication rounds, where f is the number of peers actually disrupting the protocol execution, and c is the number of rounds of the confirmation subprotocol. In the case $c=1$ such as in our Bitcoin mixing protocol (Section VI), DiceMix needs just $4+2f$ rounds.

The communication costs per run and per user are dominated by the broadcast of the DC-net array $\text{DC}[my][\,]$ of size $n\cdot|m|$ bits, where n is the number of peers and $|m|$ is the length of a mixed message. All three other broadcasts have constant size at any given security level.

A. Prototype Implementation

We have developed a proof-of-concept implementation of DiceMix based on an existing implementation of the Dissent protocol [17]. This unoptimzed implementation encompasses the complete functionality to enable testing a successful run of DiceMix without disruptions.

The implementation is written in Python and uses OpenSSL for ECDSA signatures on the `secp256k1` elliptic curve (as used in Bitcoin) at a security level of 128 bits. We use a Python wrapper for the Pari-gp library [33] to find the roots of the power sum polynomial.

1) Testbed: We tested our DiceMix implementation in Emulab [52]. Emulab is a testbed for distributed systems that enables a controlled environment with easily configurable parameters such as network topology or bandwidth of the communication links. We simulated a network setting in which all peers (10 Mbit/s) have pre-established TCP connections to a bulletin board (1 Gbit/s); all links had a delay of 50 ms. We used different Emulab machines (2.2–3.0 GHz) to simulate the peers; note that the slowest machine is the bottleneck due to the synchronization enforced by the broadcasts.

We ran the protocol with a varying number of peers, ranging from 5 to 50. Each peer had as input for the mixing a 160-bit message (e.g., a Bitcoin address).

2) Results: First, we measured wall-clock time, averaged over all peers. As shown in Fig. 4, we observe that even with 50 participants, DiceMix runs in less than 8 seconds.

Second, we measured computation time; the results are depicted in Fig. 5. We considered the average total computation time spent by a peer (purple line), and the average computation time excluding solving the equation system involving the power sums (i.e., green line).

3) Optimization: We observe that solving the equation system is quite expensive, namely about 2 seconds for 50 peers. To demonstrate that this is mostly due to lack of optimization, we developed an optimized stand-alone application for this step in C++ using the FLINT number theory library [29], which provides a highly optimized implementation of the Kaltofen-Shoup algorithm for polynomial factorization over finite fields [32]. Our optimized application solves the equation system involving the power sums in about 0.15 seconds for

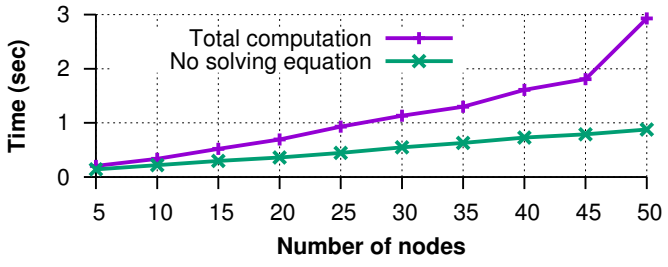

Fig. 5: Total computation time (average over peers). The purple line shows the total computation time. The green line shows the computation time without solving the equation system involving the power sums.

50 peers on a 2.10 GHz (Intel Core i7-4600U) machine, using 6 MB of DDR3-1600 RAM. This shows that optimizations can reduce the running time of the protocol further.

4) Conclusion: The experimental results show that even our unoptimized implementation of DiceMix scales to a large number of peers and outperforms state-of-the-art P2P mixing solutions such as CoinShuffle [45] and Dissent [17] considerably. In comparison, CoinShuffle (as an optimized variant of the Dissent shuffle protocol) needs slightly less than 3 minutes to complete a successful run of the P2P mixing protocol in a very similar test environment with 50 peers.

VI. Efficient Coin Mixing in Bitcoin

Several different heuristics to link Bitcoin payments sent or received by a particular user have been proposed in the literature [1], [3], [34], [40], [44], [46]. Ultimately, crypto-currencies such as Bitcoin using a public Blockchain may in fact provide *less* anonymity than traditional banking. Coin mixing has emerged as a technique to overcome this problem while maintaining full compatibility with current Bitcoin protocol.

A promising solution in this direction is CoinShuffle [45], a P2P mixing protocol based on a mixnet run by the peers to ensure the unlinkability of input and output accounts in a jointly created mixing transaction (a so-called CoinJoin transaction [37]). However, a run with a decent anonymity set of $n = 50$ peers takes about three minutes to complete [45], *assuming that every peer is honest*. In the presence of f disruptive peers aiming at impeding the protocol, $O(nf)$ communications rounds are required, and most of them inevitably taking longer due to the disruptive peers delaying their messages intentionally. For say $f = 10$ disrupting peers, the protocol needs more than 30 minutes to succeed, which arguably prohibits a practical deployment of CoinShuffle. As a consequence, we lack a coin mixing protocol for crypto-currencies that is efficient enough for practical deployment.

We propose CoinShuffle++, a highly efficient coin mixing protocol resulting from the application of DiceMix to the Bitcoin setting. In the following, we first give some background on Bitcoin. Then, we describe additional design goals when using P2P mixing protocol in payment systems, and then describe in detail our protocol. Finally, we show the experimental results from our evaluation and finally compare CoinShuffle++ with other available proposals to overcome the anonymity problem in Bitcoin.

A. The Bitcoin System

Bitcoin [42] is a crypto-currency run by a P2P network. An account in the Bitcoin system is associated with an ECDSA key pair; accounts are publicly identified by a 160-bit hash of the verification key, called *address*. Every peer can create an arbitrary number of accounts by creating fresh key pairs.

A peer can spend coins stored at her account by creating and signing Bitcoin transactions. In its simplest form, a Bitcoin transaction is composed by an input account, an output account and the amount of coins to be transferred from the input to the output account.[5] For the transaction to be successful, it must be signed with the signing key associated to the input account.

Bitcoin transactions can include multiple input and output accounts to spend coins simultaneously. In this case, the transaction must be signed with the signing keys associated to each of the input accounts.

B. Security Goals

Apart from the general security goals for a P2P mixing protocol (see Section II-F), the protocol must guarantee *correct balance*, a security property of interest when the P2P mixing protocol is leveraged to mix output accounts that enable privacy preserving transactions in payment systems.

a) Correct Balance: If the P2P mixing protocol succeeds for peer p, then balance of p's output account is at least β (ignoring transaction fees), where β is mixing amount in the P2P mixing protocol. In negative case, the total balance of the accounts of p is not reduced (ignoring transaction fees).

C. The CoinShuffle++ Protocol

CoinShuffle++ leverages DiceMix to perform a Bitcoin transaction where the input and output accounts for any given (honest) peer cannot be linked. In particular, CoinShuffle++ creates a fresh pair of signing-verification Bitcoin keys and returns the verification key to implement $\textsc{Gen}()$.

Then, for the confirmation subprotocol $\textsc{Confirm}(\ldots)$, CoinShuffle++ uses CoinJoin [37], [39] to perform the actual mixing. A CoinJoin transaction allows a set of peers to mix their coins without the help of a third party. In such a transaction, peers set their current Bitcoin accounts as input and a mixed list of fresh Bitcoin accounts as output. Crucially, peers can verify whether the thereby constructed transaction transfers the correct amount of coins to their fresh output account and only if all peers agree and sign the transaction, it becomes valid. So in the case of CoinShuffle++, the explicit confirmation provided by DiceMix is a list of valid signatures, one from each peer, on the CoinJoin transaction.

Note that DiceMix guarantees that everybody receives the correct list of outputs accounts in the confirmation subprotocol. So a peer refusing to sign the CoinJoin transaction can safely be considered malicious and removed. This is a crucial property for an anonymous CoinJoin-based approach, otherwise a single malicious peer can refuse to sign the transaction and thus mount a DoS service on all other peers who cannot exclude the malicious peer if not convinced of his guilt.

We define CoinShuffle++ in Algorithm 3. There, we denote by $\mathsf{CoinJoinTx}(VK_{in}[], VK_{out}[], \beta)$ a CoinJoin transaction that transfers β bitcoins from every input to every output account (where β is a pre-arranged parameter). Moreover, we denote by $\mathsf{Submit}(tx, \sigma[])$ the submission of tx including all signatures to the Bitcoin network.

[5] Technically, there is no notion of an input account but most of the inputs can be associated with addresses (accounts). We ignore those details because they are not relevant to our discussion.

1) Security Analysis: Observe that CoinShuffle++ adheres to the requirements specified in Section IV-B. Thus, sender anonymity and termination in CoinShuffle++ are immediate. (We refer to [39] for a detailed taint-based analysis on the privacy implications of CoinJoin-based protocol providing sender anonymity.) Correct balance is enforced by the CoinJoin paradigm: by construction, a peer signs only transactions that will transfer his funds from her input address to her output address.

2) Performance Analysis: In our performance analysis of DiceMix (Section V), GEN() creates a new ECDSA key pair and CONFIRM(...) obtains ECDSA signatures from all peers (using their initial ECDSA key pairs) on a bitstring of 160 bits. This is almost exactly CoinShuffle++, so the performance analyses of DiceMix carries over to CoinShuffle++.

3) Practical Considerations: There are several considerations when deploying CoinShuffle++ in practice. First, Bitcoin charges transactions with a small *fee* to prevent DoS attacks. Second, the mixing amount β must be the same for all peers but peers typically do not hold the exact mixing amount in their input Bitcoin account. Finally, after honestly performing the CoinShuffle++ protocol, a peer could spend her bitcoins in the input account before the CoinJoin transaction is confirmed, in an attempt of double-spending. All these challenges are easy to overcome. We refer the reader to the literature on CoinJoin based mixing, e.g., [37], [39], [45], for details.

a) Compatibility and Extensibility: Since CoinJoin transactions work in the current Bitcoin network, CoinShuffle++ is immediately deployable without any change to the system.

Moreover, the fact that DiceMix is generic in the CONFIRM(...) function allows to define variants of CoinShuffle++ to support a wide range of crypto-currencies and signature algorithms, including interactive signature protocols. cryptocurrencies.

For example, the integration of Schnorr signatures is planned in an upcoming Bitcoin software release [8]. This modification will enable aggregate signatures using a interactive two-round protocol among the peers in a CoinJoin transaction [38]. Given that signatures are often the largest individual part of the transactions, this enhancement greatly reduces the size of transactions and thus the transaction fee, thereby making mixing using CoinJoin transactions even cheaper.

Algorithm 3 CoinShuffle++

procedure GEN( )
  $(vk, sk) :=$ AccountGen() ▷ Stores sk in the wallet
  return vk

procedure CONFIRM($i, P, my, \text{VK}_{in}[\,], sk_{in}, \text{VK}_{out}[\,], sid$)
  $tx :=$ CoinJoinTx($VK_{in}[], VK_{out}[], \beta$)
  $\sigma[my] :=$ Sign(sk_{in}, tx)
  broadcast $\sigma[my]$
  receive $\sigma[\,]$ **from all** $p \in P$
    where Verify($\text{VK}_{in}[p], \sigma[p], tx$)
  missing P_{off} **do** ▷ Peers refusing to sign are malicious
    return P_{off}
  Submit($tx, \sigma[\,]$)
  return $\emptyset$ ▷ Success!

VII. RELATED WORK IN CRYPTO-CURRENCIES

In the following we overview the literature on privacy-preserving protocols for crypto-currencies. Related work for P2P mixing protocols is discussed throughout the paper.

A. Privacy-preserving Crypto-currencies

Zerocoin [41] and its follow-up work Zerocash [4], whose implementation Zcash is currently in an alpha stage [54], 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 due to the use of zkSNARKS.

CoinShuffle++ builds on top of Bitcoin, and thus can be deployed immediately and seamlessly without requiring any changes to the Bitcoin protocol. Bitcoin is by far the most widespread crypto-currency and will most probably retain this status in the foreseeable future, so users are in need of solutions enhancing privacy in Bitcoin.

The CryptoNote design [51] relies on ring signatures to provide anonymity for the sender of a transaction. In contrast to CoinShuffle++, an online mixing protocol is not necessary and a sufficient anomyity set can be created using funds of users currently not online. However, this comes with two important drawbacks for scalability.

First, CryptoNote requires each transaction to contain a ring signature of size $O(n)$, where n is the size anonymity set, whereas our approach based on CoinJoin needs only constant space per user. Storing the ring signatures requires a lot of precious space in the blockchain, and verifiying them puts a large burden on all nodes in the currency network. (In other words, the advantage of CoinShuffle++ is that it moves the anonymization work to an online mixing protocol, which is independent of the blockchain.)

Second, CryptoNote is not compatible with pruning, a feature supported e.g., by the Bitcoin Core client [7]. 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 CoinShuffle++ does not suffer from this problem and is compatible with pruning.

B. Centralized Mixing Services

Centralized mixing services [9] can be used to unlink a bitcoin from the bitcoin's owner: several owners transfer their coins to the mixing service who returns it to the owners at a fresh addresses. The main advantage of a centralized approach is that it scales well to a large anonymity sets. However, by using these services naively, a user must fully trust the mix: First, anonymity is restricted towards external observers, i.e., the mixing service itself can still determine the owner of a bitcoin. Second and even more important, the users have to transfer their funds to the mixing service who could just steal them by refusing to give them back.

Mixcoin [10] does not solve the first problem while it mitigates the second problem by holding the mix accountable in

case it steals the coins (but theft is still possible). Blindcoin [50] improves upon Mixcoin on that the mix does not learn the owner of a bitcoin. Moreover, the mix is also held accountable in case it steals the coins, but theft of coins is still possible.

Blindly Signed Contracts [31] proposes a centralized mechanism based on the combination of blind signatures and smart contract to solve both mentioned challenges, i.e., theft and anonymity. However, the adoption of this approach requires a protocol change, which can be implemented as a soft-fork in the current Bitcoin blockchain. Moreover, this mechanism requires four Bitcoin transactions per peer, three of them to be confirmed sequentially. Even when using potentially risky one-block confirmations, this implies that mixing takes 30 minutes on average and transaction fees for four transactions per peer. The follow-up work TumbleBit [30] proposes a mechanism fully compatible with Bitcoin but still requires at least two blocks to be confirmed sequentially.

CoinShuffle++ uses a single transaction for all peers and thus requires much less time and fees from the peers.

C. Other P2P Approaches

CoinParty [55] is a protocol where a set of mixing peers is used to mix coins from the users. In this approach, they assume that $1/3$ of the mixing parties are honest. However, this trust assumption is not in line with the Bitcoin philosophy, and much worse, it is unclear how to realize it in a P2P setting without strong identities, where Sybil attacks are easily possible. CoinShuffle++, instead, does not make any trust assumption on the mixing participants, except that there must be two honest peers, which is fundamental requirement for any protocol providing anonymity.

D. Sybil-Resistant Approaches

Xim [6] improves on its related previous work [3] in that it uses a fee-based advertisement mechanism to pair partners for mixing, and provides evidence of the agreement that can be leveraged if a party aborts. Even in the simple case of a mixing between two peers, Xim requires to publish several Bitcoin transactions in the Bitcoin blockchain, what takes on average at least 10 minutes for each transaction.

CoinShuffle++ instead requires to submit a single transaction to the Bitcoin blockchain independently on the number of peers. Moreover, although CoinShuffle++ does not prevent malicious peers from disrupting the protocol, it provides a mechanism to identify the misbehaving peer so that it can be excluded and termination is ensured.

VIII. A Deanonymization Attack on State-Of-The-Art P2P Mixing Protocols

In this section, we show a deanonymization attack on state-of-the-art P2P mixing protocol.

At the core of the problem is handling of passive disruption, i.e., peers that appear to be offline. However, it turns out that the ability to finish the protocol without a particular peer p is a serious problem for this peer's anonymity. In fact, we will describe an attack enabling a network attacker to break the anonymity of peer p!

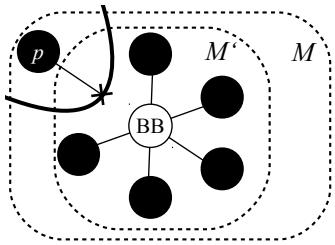

Fig. 6: A Protocol Under Attack. Peer p is partitioned from the bulletin board. Dashed rectangles indicate the message sets of the contained peers.

This is not at all a problem if peer p is proven to be an active disruptor and thus malicious. However, sacrificing the anonymity of p is a serious issue and renders any protocol insecure if p just appears to be offline. Peer p could in fact be honest, because there is no "smoking gun" that allows the other peers to conclude that p is malicious. To the best of our knowledge, this security problem has been overlooked in the literature so far.

A. Example: A Deanonymization Attack on Dissent

We exemplify the attack on the Dissent shuffle protocol [17], [49].[6] In the last communication round of the Dissent shuffle protocol, every peer publishes a decryption key. All decryption keys taken together enable the peers to decrypt anonymized ciphertexts, resulting in the final set M of anonymized messages. (The rest of the protocol is not relevant for our attack.) The attack on the shuffle protocol now proceeds as follows (Fig. 6):

1) The network attacker does not interfere with the protocol until the last communication round. In the last round the attacker partitions the network in a part with only one honest peer p and a part with the remaining peers. Consequently, the last protocol message by peer p (containing her decryption key) does not reach the other peers. As the attacker has learned all decryption keys (including that of p), he can decrypt the final set of messages M but nobody else can.[7] However, anonymity is not broken so far.
2) The remaining peers must eventually conclude that peer p is offline and exclude him; otherwise they will not be able to continue the protocol, because they cannot trust that p will ever be reachable again. The strategy by which Dissent provides termination in such a situation, is that the remaining peers will now attempt a second run the protocol without peer p. In this second run, the remaining peers *resend their messages used in the first run* [17, Section 5.4]. The attacker does not interfere with

[6]There are several protocols named Dissent. First, there is a P2P mixing protocol proposed by Corrigan-Gibbs and Ford [17] and formally proven secure by Syta et. al. [49]. Second, there is protocol [53] in a client/server setting, which requires trust in one of several servers and is consequently not relevant in context. The former (P2P) protocol by Corrigan-Gibbs and Ford [17] has two variants, a shuffle protocol and a bulk protocol. The shuffle protocol is supposed to provide anonymity but is restricted to all peers having a message of the same size, whereas the bulk protocol does not share this restriction. When we say Dissent, we always mean the shuffle protocol [17, Section 3].

[7]Dissent has the property that also a passive network observer (not participating in the protocol) can reconstruct M.

this second protocol run, and so the run will eventually succeed with a final message set M'.

3) Observe that $M' \setminus M = \{m_p\}$, since p is the only peer present in the first run but not in the second. This breaks anonymity of p.

The issue on the formal side is an arguably too weak security definition. The core of the Dissent protocol [17], [49] does not provide termination on its own but just a form of accountability, which states that at least one active disruptor can be exposed in every failed run of the protocol. The underlying idea is to use a wrapper protocol that can provide termination by starting a new run of Dissent without the exposed disruptor, after a run has failed. In this new run, the peers resend their messages [17, Section 5.4], making the whole protocol vulnerable to the attack described above.

The formal analysis of the Dissent however does not cover the wrapper protocol but considers only a single run, and correctly establishes anonymity and accountability for a single run. It has been overlooked that anonymity is lost under sequential composition of several Dissent runs using the same input messages, even though this composition is exactly what the wrapper protocol does to ensure successful termination.

Interestingly, Corrigan-Gibbs and Ford [17] acknowledge and mention the problem that the last protocol message may be withheld and thus some peer (or the network attacker) may learn the result of the protocol while denying it to others [17, Section 5.5]. However, their discussion is restricted to reliability and fails to identify the consequences for anonymity.

B. Generalizing the Attack

The underlying reason of this intersection-like attack is a fairness issue: the attacker, possibly controlling some malicious peers, can learn (parts of) the result M of an protocol execution while denying M to the other peers. If now some peer p appears offline, e.g., because the attacker blocks network messages, the remaining peers have to finish the protocol without p, resulting in a message set M', which in contrast to M does not contain m_p. Then the attacker has learned that m_p belongs to p.

Since fairness is a general problem in cryptography without honest majority, it is not surprising that the attack can be generalized. In Appendix A, we show a generic attack that breaks anonymity for any P2P mixing protocol, which provides termination and supports arbitrarily chosen input messages.

IX. Conclusions

In this work we present DiceMix, a P2P mixing protocol based on DC-nets that enable participants to anonymously publish a set of messages ensuring sender anonymity and termination. DiceMix avoids slot reservation and still ensures that no collisions occur, not even with a small probability. This results in DiceMix requiring only 4 rounds independently on the number of peers, and $4 + 2f$ rounds in the presence of f misbehaving peers. We have implemented DiceMix and showed its practicality to enable privacy preserving operations in several scenarios.

We use DiceMix to implement CoinShuffle++, a practical decentralized coin mixing for Bitcoin. Our evaluation results show that CoinShuffle++ is a promising approach to ensure sender anonymity in Bitcoin requiring no change in the current Bitcoin protocol.

References

[1] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in bitcoin," in *FC'13*.

[2] Anonymity.Online, "AN.ON," https://anon.inf.tu-dresden.de/.

[3] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better. how to make Bitcoin a better currency," in *FC'12*.

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

[5] O. Berthold, H. Federrath, and S. Köpsell, "Web mixes: A system for anonymous and unobservable internet access," in *PETS'00*.

[6] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-resistant mixing for Bitcoin," in *WPES '14*.

[7] Bitcoin Core, "0.11.0 release notes," https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md#block-file-pruning.

[8] ——, "Segregated witness: the next steps," https://bitcoincore.org/en/2016/06/24/segwit-next-steps/#schnorr-signatures.

[9] Bitcoin Wiki, "Mixing services," https://en.bitcoin.it/wiki/Category:Mixing_Services.

[10] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. Kroll, and E. Felten, "Mixcoin: Anonymity for Bitcoin with accountable mixes," in *FC'14*.

[11] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security?" in *CCS'07*.

[12] J. Bos and B. den Boer, "Detection of disrupters in the dc protocol," in *EUROCRYPT'89*.

[13] D. Cash, E. Kiltz, and V. Shoup, "The twin Diffie-Hellman problem and applications," *J. Cryptol.*, vol. 22, no. 4, 2009.

[14] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *J. Cryptol.*, vol. 1.

[15] ——, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Comm. ACM*, vol. 4, no. 2, 1981.

[16] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *S&P'15*.

[17] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable anonymous group messaging," in *CCS '10*.

[18] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security'04*.

[19] D. Dolev, R. Reischuk, and H. R. Strong, "Early stopping in byzantine agreement," *J. ACM*, vol. 37, no. 4, 1990.

[20] L. A. Dunning and R. Kresman, "Privacy preserving data sharing with anonymous id assignment," *Transactions on Information Forensics and Security*, vol. 8, no. 2, 2013.

[21] M. Florian, J. Walter, and I. Baumgart, "Sybil-resistant pseudonymization and pseudonym change without trusted third parties," in *WPES'15*.

[22] C. Franck, "Dining cryptographers with 0.924 verifiable collision resolution," arXiv CoRR abs/1402.1732, https://arxiv.org/abs/1402.1732, 2014.

[23] C. Franck and J. van de Graaf, "Dining cryptographers are practical," arXiv CoRR abs/1402.2269, https://arxiv.org/abs/1402.2269, 2014.

[24] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson, "Non-interactive key exchange," in *PKC'13*, 2013.

[25] S. Goel, M. Robson, M. Polte, and E. G. Sirer, "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication," Cornell University, Tech. Rep. 2003-1890.

[26] D. M. Goldschlag, M. Reed, and P. Syverson, "Hiding Routing Information," in *Information Hiding: First International Workshop*, 1996.

[27] P. Golle and A. Juels, "Dining cryptographers revisited," in *EUROCRYPT'04*.

[28] H. W. Gould, "The Girard-Waring power sum formulas for symmetric functions and Fibonacci sequences," *Fibonacci Quarterly*, vol. 37, no. 2, 1999, http://www.fq.math.ca/Issues/37-2.pdf.

[29] W. Hart, F. Johansson, and S. Pancratz, "FLINT: Fast Library for Number Theory," 2015, version 2.5.2, http://flintlib.org.

[30] E. Heilman, F. Baldimtsi, L. Alshenibr, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted tumbler for Bitcoin-compatible anonymous payments," IACR Cryptology ePrint 2016/575. https://eprint.iacr.org/2016/575.

[31] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain Bitcoin transactions," IACR Cryptology ePrint 2016/056. https://eprint.iacr.org/2016/056.

[32] E. Kaltofen and V. Shoup, "Fast polynomial factorization over high algebraic extensions of finite fields," in *ISSAC'97*.

[33] klinck (pseudonym), "PARI-Python interface," https://code.google.com/archive/p/pari-python/.

[34] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in Bitcoin using P2P network traffic," in *FC'14*, 2014.

[35] A. Krasnova, M. Neikes, and P. Schwabe, "Footprint scheduling for dining-cryptographer networks," in *FC'16*.

[36] D. Krawisz, "Mycelium Shufflepuff (an inplementation of CoinShuffle)," https://github.com/DanielKrawisz/Shufflepuff.

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

[38] ——, "Signature aggregation for improved scalablity," 2016, https://bitcointalk.org/index.php?topic=1377298.0.

[39] S. Meiklejohn and C. Orlandi, "Privacy-enhancing overlays in Bitcoin," in *BITCOIN'15*.

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

[41] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from Bitcoin," in *S&P'13*.

[42] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, 2008.

[43] NXT, "1.7 release," http://www.nxtinfo.org/2015/11/30/nxts-upcoming-1-7-release-featuring-coin-shuffling-singleton-assets-account-control-and-an-improved-forging-algorithm/.

[44] F. Reid and M. Harrigan, "An analysis of anonymity in the Bitcoin system," in *SXSW'13*.

[45] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "CoinShuffle: Practical decentralized coin mixing for Bitcoin," in *ESORICS'14*, 2014.

[46] M. Spagnuolo, F. Maggi, and S. Zanero, "BitIodine: Extracting intelligence from the Bitcoin network," in *FC'14*, 2014.

[47] T. K. Srikanth and S. Toueg, "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms," *Distributed Computing*, vol. 2, no. 2, 1987.

[48] K. F. Stphan Kochen, Alexey Sokolov, "IRCv3.2 server-time extension," 2012, http://ircv3.net/specs/extensions/server-time-3.2.html.

[49] E. Syta, H. Corrigan-Gibbs, S.-C. Weng, D. Wolinsky, B. Ford, and A. Johnson, "Security analysis of accountable anonymity in Dissent," *TISSEC*, vol. 17, no. 1, 2014.

[50] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for Bitcoin," in *BITCOIN'15*.

[51] N. van Saberhagen, "Cryptonote," 2013, https://cryptonote.org/whitepaper.pdf.

[52] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, 2002.

[53] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in numbers: Making strong anonymity scale," in *OSDI'12*, 2012.

[54] Zcash, "New alpha release: libzcash," https://z.cash/blog/new-alpha-release-libzcash.html.

[55] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "CoinParty: Secure multi-party mixing of bitcoins," in *CODASPY'15*.

APPENDIX

A. Generic Attack

In this section, we generalize the attack described in Section VIII. The resulting attack breaks anonymity on every P2P mixing protocol, which supports arbitrary input messages and is supposed to ensure termination.

We assume an execution of a P2P mixing protocol with peer set $P = \{p_1, \ldots, p_n\}$ and their set of fixed input messages $M = \{m_1, \ldots, m_n\}$. We further assume that the attacker controls the network and a majority $A \subset P$ of peers in the execution such that $|P|/2 < |A| \leq |P| - 3$.

For the sake of presentation, we assume that no two peers send a protocol message in the same communication round. (This models that the network attacker can determine the order of simultaneous messages arbitrarily.)

Let r be the first communication round (or protocol message) after which input message m_i of peer p_i is known to a collusion of a minority S of peers with $p_i \notin S$.[8] Such a round must exist, because every peer is supposed to output M at the end of successful protocol execution, and M contains m_i. Note that knowledge of m_i does not imply that the collusion S of peers collectively knows that m_i belongs to peer p_i, it just means that the collusion knows that the bitstring m_i is *one* of the peers' input messages.

Assume that $S \subset A$, i.e., S is entirely controlled by the attacker; this happens with non-negligible probability if the attacker randomly selects malicious peers in the beginning.

The attacker lets the protocol run until round r and learns m_i (by control of S). Then the attacker uniformly selects an index i^* from the set of honest peers. From round r on, the attacker blocks all further protocol messages by p_{i^*}, and by his own peers in A; all these peers will appear offline for the remaining peers in $R := P \setminus (\{p_{i^*}\} \cup A)$. By assumption, $|R| \geq 2$. Hence by the termination property, those remaining peers in R finish the protocol with a result set $M' \subset M$.

Observe that the remaining peers form a minority. If $i^* = i$, then p_i is not among them. As a minority, they do not know m_i; recall that we have chosen r in such a way that no minority knows m_i. Thus $m_i \notin M'$. If however $i^* \neq i$, then p_i is among them, and correctness of the protocol implies $m_i \in M'$. In other words, the attacker learns whether m_i belongs to peer p_{i^*} or not. This breaks the anonymity of p_{i^*}.

[8] Formally, this is the first round r for which an efficient extraction algorithm E exists such that E outputs m_i with non-negligible probability, given the full state of all peers in S after round r.