

P2P Mixing and Unlinkable Bitcoin Transactions

Anonymity of the People, by the People, and for the People

Tim Ruffing
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-nets), several peer-to-peer (P2P) anonymous communication protocols have been proposed. However, despite their strong anonymity guarantees, none of them have been employed in practice so far: Most protocols fail to simultaneously address the crucial problems of slot collisions and disruption by malicious peers, while the remaining ones handle f malicious peers with $O(f^2)$ communication rounds. We conceptualize these P2P anonymous communication protocols as *P2P mixing*, and present a novel P2P mixing protocol, DiceMix, that needs only four communication rounds in the best case, and $4 + 2f$ rounds in the worst case with f malicious peers. As every individual malicious peer can force a restart of a P2P mixing protocol by simply omitting his messages, we find DiceMix with its worst-case complexity of $O(f)$ rounds to be an optimal P2P mixing solution.

On the application side, we employ DiceMix to improve anonymity in 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. We use DiceMix to define CoinShuffle++, a coin mixing protocol that enables pseudonymous peers to perform unlinkable transactions in a manner fully compatible with the current Bitcoin system. Moreover, we demonstrate the efficiency of our protocols with a proof-of-concept implementation. In our evaluation, DiceMix requires less than eight seconds to mix 50 messages (160 bits, i.e., Bitcoin addresses), while the best protocol in the literature requires almost three minutes in the same setting.

Finally, we present a deanonymization attack on existing P2P mixing protocols that guarantee termination in the presence of disruptive peers. We generalize the attack to demonstrate that no P2P mixing protocol simultaneously supports arbitrary input messages, provides anonymity, and terminates in the presence of disruptive peers. DiceMix resists this attack by requiring fresh input messages, e.g., cryptographic keys never used before.

I. INTRODUCTION

Chaum [18] introduced the concept of anonymous digital communication in the form of mixing networks (or *mixnets*). In the mixnet protocol, a batch of encrypted messages from users is decrypted, randomly permuted, and relayed by a sequence of

routers to avoid individual messages getting traced through the network. The original mixnet protocol, as well as its successors such as onion routing [31], AN.ON [1], and Tor [23], inherently require access to a set of geographically distributed routers such that at least some of them are trusted to not break peers' anonymity.

Starting with the Dining Cryptographers network (DC-net) protocol [17], another line of research on anonymous communication networks emerged, in which peers do not depend on any third-party routers and instead communicate with each other to send their messages anonymously. While the DC-net protocol can guarantee successful termination and anonymity against an honest-but-curious adversary controlling a subset of peers, it is prone to disruption by a single malicious peer who sends invalid protocol messages (active disruption), or simply omits protocol messages entirely (passive disruption). Moreover, a DC-net protects the anonymity of the involved malicious peers, making it impossible for honest peers to detect and exclude the malicious peer.

To address this termination issue, recent successors of the DC-net protocol [15], [22], [27], [28], [32], [55] incorporate cryptographic accountability mechanisms against active disruptions. The employed techniques are either proactive, e.g., zero-knowledge proofs proving the validity of sent messages [32], or reactive, e.g., the revelation of session secrets to expose and exclude malicious disruptors after a failed protocol run [22]. These protocols 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. Moreover, given the surging demand for anonymity in P2P crypto-currencies such as Bitcoin [5], [6], [40], [45], [46], [51], [53], these protocols have led to real-world P2P Bitcoin mixing systems [3], [42], [52].

Nevertheless, these solutions are still not ideal: with communication rounds quadratic in the worst case with many malicious peers, these current pure P2P solutions [22], [52], [55] do not scale as the number of participating peers grows. For instance, the state-of-the-art Bitcoin P2P mixing protocol CoinShuffle [52] requires a few minutes to anonymize the communication of 50 peers if every peer is honest, and even much longer in the presence of malicious peers. In this paper, it is our goal to bring P2P anonymous communication from the realm of feasibility to the realm of practicality.

A. Contributions

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

1) *P2P Mixing*: As our first contribution, we conceptualize P2P mixing as a natural generalization of DC-nets [17]. A P2P mixing protocol enables a set of mutually distrusting peers to publish their messages simultaneously and anonymously without requiring any trusted or untrusted third-party anonymity proxy.

2) *DiceMix Protocol*: Although some DC-net successors [30], [41] as well as some anonymous group messaging systems [22], [52], [55] satisfy the P2P mixing requirements, we found those to be too inefficient for large-scale mixing. As our second contribution, we present the new P2P mixing protocol DiceMix, which builds on the original DC-net protocol. P2P Mixing Protocol handles collisions by redundancy, and disruption by revealing session secrets to expose malicious peers. DiceMix requires only $4+2f$ rounds in the presence of f malicious peers, i.e., only four rounds if every peer behaves honestly. The resulting communication round complexity is a linear factor better than in existing state-of-the-art approaches [22], [30], [41], [52].

We provide a proof-of-concept implementation of the DiceMix protocol, and evaluate it in Emulab [59]. Our results show that in an Internet-like setting, 50 peers can anonymously broadcast their messages in about eight seconds, whereas previous state-of-the-art protocols need several minutes.

3) *CoinShuffle++ Protocol*: As our third contribution, we apply DiceMix to Bitcoin, the most popular crypto-currency. In particular, building on the CoinJoin paradigm [43] and DiceMix, we present CoinShuffle++, a practical decentralized mixing protocol for Bitcoin users. CoinShuffle++ not only is considerably simpler and thus easier to implement than its predecessor CoinShuffle [52] but also inherits the efficiency of DiceMix and thus outperforms CoinShuffle significantly. In particular, in a scenario with 50 participants in the same evaluation setting, a successful transaction with CoinShuffle++ can be created in eight seconds, instead of the almost three minutes required with CoinShuffle.

4) *A Generic Attack on P2P Mixing Protocols*: As our fourth contribution, we present a deanonymization attack on existing P2P mixing protocols that guarantee termination in the presence of disruptive peers. We exemplify the attack on the Dissent shuffle protocol [22], [55] and then generalize the attack to demonstrate that no P2P mixing protocol simultaneously supports arbitrary input messages, provides anonymity, and terminates in the presence of disruptive peers.

The proposed attack is similar to statistical disclosure attacks across several protocol runs (e.g., [14], [61]) but works with certainty, because a protocol, which is supposed to terminate successfully, can be forced to start a new run to ensure termination. Finally, we discuss how DiceMix resists this attack by requiring fresh input messages (e.g., cryptographic keys never used before), and we discuss why this is not a problem for applications such as coin mixing.

II. CONCEPTUALIZING P2P MIXING

A P2P mixing protocol [22], [52], [62] allows a group of mutually distrusting peers, each having an input message, to simultaneously broadcast their messages in an anonymous manner *without the help of a third-party anonymity proxy*. An

attacker controlling the network and some peers should not be able to tell which of the 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 anonymity 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 a meaningful anonymity guarantee in the presence of an attacker controlling all but two peers. As a consequence, each peer must be actively involved in the anonymous communication process which comes with inherent restrictions and expense.

A. 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; the bulletin board is purely a matter of communication.

We assume the bounded synchronous communication setting, where time is divided into 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 be seamlessly deployed in practice, and in fact already-deployed Internet Relay Chat (IRC) servers suffice.¹ The bulletin board can alternatively be substituted by an (early stopping) reliable broadcast protocol [24], [54] if one is willing to accept the increased communication cost.

We assume that all peers participating 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 is necessary. Note that a malicious bootstrapping mechanism may 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, decreasing the size of her effective anonymity set. 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.

B. 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 all but two peers

¹ Servers supporting IRC version 3.2 are capable of adding a server timestamp to every message [39]; this can ensure that peers agree whether a certain message arrived in time.

introduces an unexpected requirement, namely, that 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 protocol 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 [22] and the protocol by Golle and Juels [32], we require that input messages to be mixed are freshly drawn from a distribution with sufficient entropy, e.g., input messages can be random bitstrings or public keys never used before. 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 them.

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 instead peers can arbitrarily choose their messages in a P2P mixing protocol guaranteeing termination, the protocol is inherently vulnerable to an attack breaking sender anonymity. We will explain this attack in detail in Section VIII; it works against state-of-the-art P2P mixing protocols and has been overlooked in this form in the literature so far.

2) *Explicit Confirmation of the Output*: Anonymity-seeking P2P applications such as coin mixing [43], [52], [62] or identity mixing [26] 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 unexcluded peers.

C. 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 $\text{GEN}()$ to obtain a fresh input message to be mixed. If a run is disrupted, the protocol can exclude 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* $\text{CONFIRM}(i, P, M)$, 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 threshold signature in a distributed fashion.

If confirmation can be obtained from everybody, then the run and the P2P mixing protocol *terminates successfully*. Otherwise, $\text{CONFIRM}(i, P, M)$ by convention fails and reports the 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 $\text{GEN}()$; 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.

D. Threat Model

In general, we assume that the attacker controls some number f of n 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 messages from honest peers. In the case of successful termination, the anonymity set of each honest peer will be the set of unexcluded honest peers². 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, we trust the bulletin board to relay messages reliably and honestly, because termination (or any liveness property) is impossible to achieve against a malicious bulletin board, which can just block all communication.

E. Security Goals

A P2P mixing protocol must provide two security properties.

a) *Sender Anonymity*: If the protocol succeeds for honest peer p in a run (as described in Section II-C) 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' .

b) *Termination*: If the bulletin board is honest and there are at least two honest peers, the protocol eventually terminates successfully for every honest peer.

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-C). This demands explanation, because giving up anonymity in the case of failed confirmation seems to put privacy at risk at first glance. However, the discarded messages are just randomly generated bitstrings and have never been and will never be used outside the P2P mixing protocol; in

²A 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.

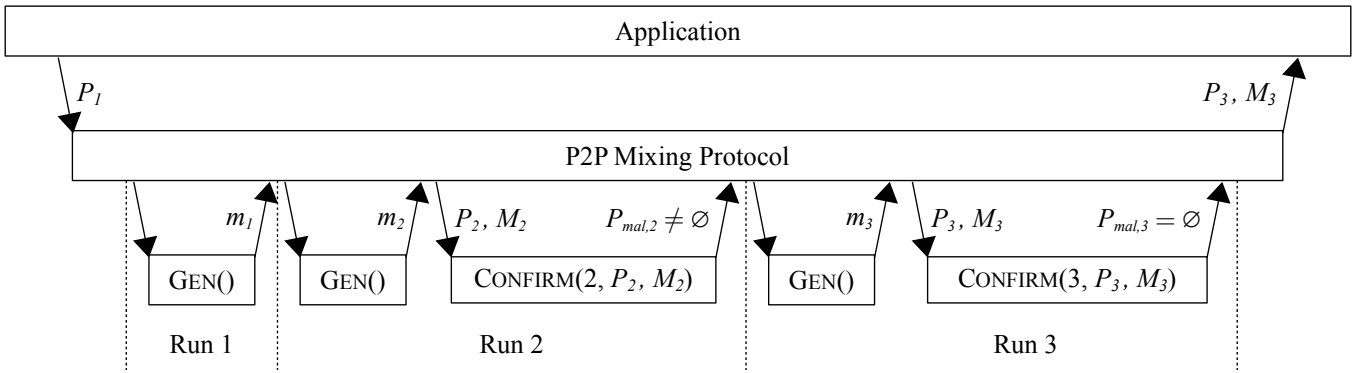


Fig. 1: Example Execution of a P2P Mixing Protocol. The figure shows the calls during the execution; time proceeds from left to right. The execution starts with the application calling the P2P mixing protocol with an initial set P_1 of peers. The P2P mixing protocol then starts Run 1 by generating a new message m_1 (via calling $\text{GEN}()$). 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 $P_{\text{mal},2}$ 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 $P_3 = P_2 \setminus P_{\text{mal},2}$. This time, the confirmation subprotocol succeeds and indicates this 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.

particular the messages have been not returned back to the application. 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.

III. SOLUTION OVERVIEW

Our core tool to design an efficient P2P mixing protocol is a Dining Cryptographers network (DC-net) [17]: Suppose that each pair of peers (i, j) shares a symmetric key $k_{i,j}$ and that one of the peers (e.g., p_1) wishes to anonymously publish a message m such that $|m| = |k_{i,j}|$. For that, p_1 publishes $M_1 := m \oplus k_{1,2} \oplus k_{1,3}$, p_2 publishes $M_2 := k_{1,2} \oplus k_{2,3}$ and finally p_3 publishes $M_3 := k_{1,3} \oplus k_{2,3}$. Now, the peers (and observers) can compute $M_1 \oplus M_2 \oplus M_3$, effectively recovering m . However, the origin of the message m is hidden: without knowing the secrets $k_{i,j}$, no observer can determine which peer published m . Additionally, the origin is also hidden for peers themselves (e.g., as p_2 does not know $k_{1,3}$, she cannot discover whether p_1 or p_3 is the origin of the message). It is easy to extend this basic protocol to more users [32].

Besides the need for pairwise symmetric keys, which can be overcome by a key exchange mechanism, there are two key issues to overcome, 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 (called *slots*) of a DC-net, where each peer randomly selects one 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 then unrecoverable [32].

One proposed solution is to perform an anonymous reservation mechanism so that peers agree in advance on a slot assignment for publishing [30], [41]. However, this mechanism adds communication rounds among the peers and it must also provide anonymity, which typically makes it prone to the same issues (e.g., slot collisions) that we would like to overcome in the first place. Alternatively, it is possible to establish many more slots so that the probability of a collision decreases [21]. 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 [15], [19], [25], [38], [50]. Assume that messages to be mixed are encoded as elements of a finite field \mathbb{F} with $|\mathbb{F}| > n$, where n is the number 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 yields an intentional collision involving all peers in each of the slots. Using addition in \mathbb{F} instead of XOR to create DC-net messages, the j -th slot contains the power sum $S_j = \sum_i m_i^j$.

Now, we require a mechanism to extract the messages m_j from the power sums S_j . Let $g(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ be a polynomial with roots m_1, m_2, \dots, m_n . Newton's identities [33] state

$$\begin{aligned} a_n &= 1, \\ a_{n-1} &= S_1, \\ a_{n-2} &= (a_{n-1} S_1 - S_2)/2, \\ a_{n-3} &= (a_{n-2} S_1 - a_{n-1} S_2 + S_3)/3, \\ &\vdots \end{aligned}$$

By knowledge of all coefficients a_j of the polynomial g , we can find its n roots, which are the n input messages.

B. Handling Disruption and Ensuring Termination

Recovering the messages only works when all peers honestly follow the protocol. If a malicious peer disrupts the DC-net by simply using inconsistent DC-net messages, we must ensure that the protocol still terminates eventually.

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 will all honest peers agree on whether disruption has occurred and are able to 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 she sees the vectors of other peers. In this manner, malicious peers are forced to create their DC-net vectors independently of the input messages of honest peers. The redundant encoding of messages using powers 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 messages 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, 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 ephemeral 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 further runs.

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 supposed to ensure termination [22], [52], [55].

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 (KeyGen, Sign, Verify) unforgeable under chosen-message attacks (UF-CMA). The algorithm KeyGen returns a private signing key sk and the corresponding public verification key vk . On input message m , Sign(sk, m) returns σ , a signature on message m using signing key sk . The verification algorithm Verify(pk, σ, 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 (NIKE.KeyGen, NIKE.SharedKey) secure in the CKS model [16], [29]. The algorithm NIKE.KeyGen(id) outputs a public key npk and a secret key nsk for a given party identifier id . NIKE.SharedKey($id_1, id_2, nsk_1, npk_2, sid$) outputs a shared key for the two parties id_1 and id_2 and session identifier sid . NIKE.SharedKey must fulfill the standard correctness requirement that for all session identifiers sid , all parties id_1, id_2 , and all corresponding key pairs (npk_1, nsk_1) and (npk_2, nsk_2) , it holds that NIKE.SharedKey($id_1, id_2, nsk_1, npk_2, sid$) = NIKE.SharedKey($id_2, id_1, nsk_2, npk_1, sid$). Additionally, we require an algorithm NIKE.ValidatePK(npk), which outputs *true* iff npk is a public key in the output space of NIKE.KeyGen, and we require an algorithm NIKE.ValidateKeyPair(npk, nsk) which outputs *true* iff nsk is a valid secret key for the public key npk .

Static Diffie-Hellman key exchange satisfies these requirements [16], given a suitable key derivation algorithm such as NIKE.SharedKey(id_1, id_2, x, g^y) := K($((g^{xy}, \{id_1, id_2\}, sid)$) for a hash function K modeled as a random oracle.

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 $\text{ARR}[i]$, where i is the index. We denote the full array (all its elements) as $\text{ARR}[]$.

A protocol message msg is broadcast using the instruction “**broadcast** m ”. The instruction “**receive** $\text{ARR}[p]$ **from all** $p \in P$ **where** $X(\text{ARR}[p])$ **missing** $C(P_{\text{off}})$ ” attempts to receive a message from all peers $p \in P$. The first message msg from peer p that fulfills predicate $X(msg)$ is accepted and stored as $\text{ARR}[p]$; all other messages from p are ignored. When a timeout is reached, the command C is executed, which has access to a set $P_{\text{off}} \subseteq P$ of peers that have not sent a (valid) message.

Regarding concurrency, a thread t that runs a procedure $P(args)$ is started using “ $t := \text{fork } P(args)$ ”. A thread with handle t can either be joined using “ $r := \text{join } t$ ”, 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 wake up thread t and notify it of value v .

B. Contract with the Application

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

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

Fig. 2: Example of 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., when a run notifies 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.

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 refuses confirmation for a legitimate reason, e.g., an incorrect set final set M not containing her message, our protocol ensures that all honest peers deliver the same and correct message set M . Then, the confirmation subprotocol $\text{CONFIRM}(i, P, M)$ can safely assume that peers refusing to confirm are malicious. We call this property *validity*.

The purpose of both of these guarantees is to ensure correct functionality of the confirmation subprotocol, 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. The guarantees are detailed below.

a) *Agreement:* Assume that the bulletin board is honest. Let p and p' be two honest peers in a protocol execution. If p calls $\text{CONFIRM}(i, P, M)$ ³ in some communication round r , then p' calls $\text{CONFIRM}(i, P, M)$ 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 $\text{CONFIRM}(i, P, M)$ 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 $|M| \leq |P|$.

2) *Requirements of 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 $\text{GEN}()$ are encoded in a prime field \mathbb{F}_q , where q is larger than the number of peers in the protocol. Also, we assume that the message m returned by $\text{GEN}()$ has sufficient entropy such that it can be predicted only with negligible probability, which also implies that q is at least as large as the security parameter.

³ $\text{CONFIRM}()$ will actually take more arguments, but they are not relevant for this subsection.

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 honest 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 $\text{CONFIRM}(i, P, M)$ succeeds for peer p (i.e., if the call returns an empty set $P_{\text{mal}} = \emptyset$ of malicious peers refusing confirmation), then all honest peers in P have called $\text{CONFIRM}(i, P, M)$.

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

C. Protocol Description

We describe the DiceMix protocol in Algorithm 1. The black code is the basic part of the protocol; the blue code handles concurrent runs and offline peers.

1) *Single Run of the Protocol (Black Pseudocode):* The protocol starts in $\text{DICE MIX}()$, which takes as input a set of other peers P , the peer's own identity my , an array $\text{VK}[]$ of verification keys of all peers, the peer's own signing key sk , and a predetermined unique session identifier sid . A single protocol run, implemented in $\text{RUN}()$, consists of four rounds.

In the first round (KE), the NIKÉ is used to establish pairwise symmetric keys between all peers (DC-KEYS()). Then each peer can derive the DC-net pads from these symmetric keys (DC-SLOT-PAD()) and use them to create the vector of messages for the DC-net (DC-MIX()). 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. In the third round (DC), the peers open their commitments. 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-MIX-RES()).

Finally, every peer checks whether her input message is in the result of the DC-net, determining how to proceed in the fourth round. 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 (CF). Otherwise, she outputs her secret key. In this case, every other peer publishes her secret key as well, and the peers can replay each other's protocol messages for the current run. This will expose the misbehaving peer, and honest peers will exclude him from the next run (SK).

⁴This 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.

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 depicted 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 round (SK or CF) before run $i + 1$ reaches the DC round.

Before its DC round, run $i + 1$ can be patched as follows. In the DC round 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 malicious peers from run $i + 1$. We note that a peer could reveal wrong symmetric keys in this step. This, however, leads to wrong output from the DC-net, which is then handled by revealing secret keys in round $i + 1$. Publishing partial symmetric keys does not compromise sender anonymity for unexcluded peers because 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 to 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).

To be able to achieve termination $4 + 2f$ in communication rounds, it is crucial that missing messages in the first two broadcasts (KE and CM) do not require aborting the run. Luckily, the current run can be continued in those cases. 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 (see the code blocks starting with the “missing” instruction).

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 of validity and agreement that the application may rely on.

1) *Sender Anonymity*: Consider a protocol execution in which an 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()`, peer p' has started `CONFIRM(i, P, M)` in the same communication round as p . By construction of the protocol, this implies two properties about peer p' : (i) p' will not reveal her secret key in round SK, and (ii) p' assumes that p is not excluded in run i , and thus has not revealed the symmetric key shared with p in round RV.

As the key exchange scheme is secure in the CKS model and the exchanged public keys are authenticated using unforgeable signatures, the attacker cannot distinguish the pads derived from the symmetric key between p and p' from random pads.

Thus, after opening the commitments on the pads, peer p has formed a proper DC-net with at least peer p' . The security guarantee of Chaum’s original DC-nets [17] 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 $|M| \leq |P|$.

For part (i) 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()` (Line 48 in Algorithm 1).

We now show as intermediate step that in every run i , the condition $m \in M$ evaluates to true for all honest peers or false for all honest peers. Note that 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 $(m_a, m_a^2, \dots, m_a^n)$ with $n \geq 3$, because there is at the least malicious peer a and two honest peers. Since $m_p \in M$, this ill-formed vector left the message m_p intact. This implies that the vector of a was chosen depending on the other DC-net vectors. A simple algebraic argument shows that even for the second power sum in the second slot, it is not feasible to come up with an additive offset to the power sum that changes some of the encoded messages but leaves all others intact: To change $m_{p'}$ to $m_{p'} + \Delta$ and leave all other messages intact, the correct offset for the first slot is Δ , and the correct offset for the second slot is $(m_{p'} + \Delta)^2 - m_{p'}^2 = 2m_{p'}\Delta + \Delta^2$, which depends on $m_{p'}$ for fixed Δ . However, it is not feasible for the attacker to create one (or more) commitments on messages that depend on $m_{p'}$, because the commitments are non-malleable

Algorithm 1 DiceMix

```

1: proc DICE MIX( $P, my, VK[], sk, sid$ )
2:    $sid := (sid, P, VK[])$ 
3:   if  $my \in P$  then
4:     fail “cannot run protocol with myself”
5:   return  $RUN(P, my, VK[], sk, sid, 0)$ 
6: proc  $RUN(P, my, VK[], sk, sid, run)$ 
7:   if  $P = \emptyset$  then
8:     fail “no honest peers”
9:    $\triangleright$  Exchange pairwise keys
10:   $(NPK[my], NSK[my]) := NIKE.KeyGen(my)$ 
11:   $sidHpre := H((sidHpre, sid, run))$ 
12:  broadcast  $(KE, NPK[my], Sign(sk, (NPK[my], sidHpre)))$ 
13:  receive  $(KE, NPK[p], \sigma[p])$  from all  $p \in P$ 
14:    where  $NIKE.ValidatePK(NPK[p])$ 
15:     $\wedge Verify(VK[p], \sigma[p], (NPK[p], sidHpre))$ 
16:  missing  $P_{off}$  do
17:     $P := P \setminus P_{off}$   $\triangleright$  Exclude offline peers
18:   $sidH := H((sidH, sid, P \cup \{my\}, NPK[], run))$ 
19:   $K[] := DC-KEYS(P, NPK[], my, NSK[my], sidH)$ 
20:   $\triangleright$  Generate fresh message to mix
21:   $m := GEN()$ 
22:   $DC[my][] := DC-MIX(P, my, K[], m)$ 
23:   $P_{ex} := \emptyset$   $\triangleright$  Malicious (or offline) peers for later exclusion
24:   $\triangleright$  Commit to DC-net vector
25:   $COM[my] := H((CM, DC[my][[]]))$ 
26:  broadcast  $(CM, COM[my], Sign(sk, (COM[my], sidH)))$ 
27:  receive  $(CM, COM[p], \sigma[p])$  from all  $p \in P$ 
28:    where  $Verify(VK[p], \sigma[p], (COM[p], sidH))$ 
29:  missing  $P_{off}$  do
30:     $P_{ex} := P_{ex} \cup P_{off}$   $\triangleright$  Store offline peers for exclusion
31:  if  $run > 0$  then
32:     $\triangleright$  Wait for prev. run to notify us of malicious peers
33:     $P_{exPrev} := wait$ 
34:     $P_{ex} := P_{ex} \cup P_{exPrev}$ 
35:   $\triangleright$  Collect shared keys with excluded peers
36:  for all  $p \in P_{ex}$  do
37:     $K_{ex}[my][p] := K[p]$ 
38:   $\triangleright$  Start next run (in case this one fails)
39:   $P := P \setminus P_{ex}$ 
40:   $next := fork RUN(P, my, VK[], sk, sid, run + 1)$ 
41:   $\triangleright$  Open commitments and keys with excluded peers
42:  broadcast  $(DC, DC[my][[]], K_{ex}[my][[]], Sign(sk, K_{ex}[my][[]]))$ 
43:  receive  $(DC, DC[p][[]], K_{ex}[p][[]], \sigma[p])$  from all  $p \in P$ 
44:    where  $H((CM, DC[p][[]])) = COM[p]$ 
45:     $\wedge \{p' : K_{ex}[p][p'] \neq \perp\} = P_{ex}$ 
46:     $\wedge Verify(VK[p], K_{ex}[p][[]], \sigma[p])$ 
47:  missing  $P_{off}$  do
48:     $\triangleright$  Abort and rely on next run
49:    return  $RESULT-OF-NEXT-RUN(P_{off}, next)$ 
50:   $M := DC-MIX-RES(P \cup \{my\}, DC[[]][[]], P_{ex}, K_{ex}[[]][[]])$ 
51:   $\triangleright$  Check if our output is contained in the result
52:  if  $m \in M$  then
53:     $P_{mal} := CONFIRM(i, P, M, my, VK[], sk, sid)$ 
54:    if  $P_{mal} = \emptyset$  then
55:       $\triangleright$  Success?
56:      abort next
57:      return  $m$ 
58:    else
59:      broadcast  $(SK, NSK[my])$   $\triangleright$  Reveal secret key
60:
61:  receive  $(SK, NSK[p])$  from all  $p \in P$ 
62:  where  $NIKE.ValidateKeyPair(NPK[p], NSK[p])$ 
63:  missing  $P_{off}$  do
64:     $\triangleright$  Abort and rely on next run
65:    return  $RESULT-OF-NEXT-RUN(P_{off}, next)$ 
66:   $\triangleright$  Determine malicious peers using the secret keys
67:   $P_{mal} := BLAME(P, NPK[], my, NSK[], DC[[]][[]], sidH$ 
68:     $, P_{ex}, K_{ex}[[]][[]])$ 
69:  return  $RESULT-OF-NEXT-RUN(P_{mal}, next)$ 
70:
71: proc  $DC-MIX(P, my, K[], m)$ 
72:   $\triangleright$  Create power sums in individual slots
73:  for  $i := 1, \dots, |P| + 1$  do
74:     $DCMY[i] := m^i + DC-SLOT-PAD(P, my, K[], i)$ 
75:  return  $DCMY[]$ 
76:
77: proc  $DC-MIX-RES(P_{all}, DCMIX[[]][[]], P_{ex}, K_{ex}[[]][[]])$ 
78:  for  $s := 1, \dots, |P_{all}|$  do
79:     $M^*[s] := DC-SLOT-OPEN(P_{all}, DCMIX[[]][[]], s, P_{ex}, K_{ex}[[]][[]])$ 
80:   $\triangleright$  Solve equation system for array  $M[]$  of messages
81:   $M[] := Solve(\forall s \in \{1, \dots, |P_{all}|\}. M^*[s] = \sum_{i=1}^{|P_{all}|} M[i]^s)$ 
82:  return  $Set(M[])$   $\triangleright$  Convert  $M[]$  to an (unordered) multiset
83:
84: proc  $DC-SLOT-PAD(P, my, K[], s)$ 
85:  return  $\sum_{p \in P} sgn(my - p) \cdot G((K[p], s))$   $\triangleright$  in  $\mathbb{F}$ 
86:
87: proc  $DC-SLOT-OPEN(P_{all}, DC[[]][[]], s, P_{ex}, K_{ex}[[]][[]])$ 
88:   $\triangleright$  Pads cancel out for honest peers
89:   $m^* := \sum_{p \in P_{all}} DC[p][s]$   $\triangleright$  in  $\mathbb{F}$ 
90:
91:   $\triangleright$  Remove pads for excluded peers
92:   $m^* := m^* - \sum_{p \in P_{ex}} DC-SLOT-PAD(P_{ex}, p, K_{ex}[[]], s)$ 
93:  return  $m^*$ 
94:
95: proc  $DC-KEYS(P, NPK[], my, nsk, sidH)$ 
96:  for all  $p \in P$  do
97:     $K[p] := NIKE.SharedKey(my, p, nsk, NPK[p], sidH)$ 
98:  return  $K[]$ 
99:
100: proc  $BLAME(P, NPK[], my, NSK[], DC[[]][[]], sidH, P_{ex}, K_{ex}[[]][[]])$ 
101:   $P_{mal} := \emptyset$ 
102:  for all  $p \in P$  do
103:     $P' := (P \cup \{my\}) \setminus \{p\}$ 
104:     $K'[] := DC-KEYS(P', NPK[], p, NSK[p], sidH)$ 
105:     $\triangleright$  Reconstruct purported message  $m'$  of  $p$ 
106:     $m' := DC[p][1] - DC-SLOT-PAD(P', p, K'[], 1)$ 
107:     $\triangleright$  Replay DC-net messages of  $p$ 
108:     $DC'[[]] := DC-MIX(P', p, K'[], m')$ 
109:    if  $DC'[[]] \neq DC[p][[]]$  then  $\triangleright$  Exclude inconsistent  $p$ 
110:       $P_{mal} := P_{mal} \cup \{p\}$ 
111:     $\triangleright$  Verify that  $p$  has published correct symmetric keys
112:  for all  $p_{ex} \in P_{ex}$  do
113:    if  $K_{ex}[p][p_{ex}] \neq K'[p_{ex}]$  then
114:       $P_{mal} := P_{mal} \cup \{p\}$ 
115:  return  $P_{mal}$ 
116:
117: proc  $RESULT-OF-NEXT-RUN(P_{exNext}, next)$ 
118:   $\triangleright$  Hand over to next run and notify of peers to exclude
119:  notify next of  $P_{exNext}$ 
120:   $\triangleright$  Return result of next run
121:   $result := join next$ 
122:  return  $result$ 

```

and $m_{p'}$ cannot be predicted. This argument can be generalized to changing exactly d messages for $1 \leq d < n$.

As the message $H((CM, DC[my][[]]))$ implements a hiding, binding and non-malleable commitment on $DC[my][[]]$ (recall that adding randomness is not necessary because there is sufficient entropy in $DC[my][[]]$), 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 part (i) of validity.

For part (ii) ($|M| \leq |P|$) 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. Thus $|M| = |P|$. This shows validity.

3) *Agreement*: To show agreement, we have to show that for each run i , if one honest peer p calls $CONFIRM(i, P, M)$ in some round, then every honest peer p' calls $CONFIRM(i, P, M)$ in the same round. This follows from validity: By part (i) of validity, we know that if some honest peer calls $CONFIRM(i, P, M)$, then $m_{p'} \in M$ for every peer p' in run i . By construction of the protocol (Line 48), the condition $m_{p'} \in M$ is exactly what determines whether p' calls $CONFIRM(i, P, M)$. Thus every honest peer p' calls $CONFIRM(i, P, M)$ in the same round, which shows agreement.

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-revealed secret keys) clearly identifies at least one malicious peer, and since all honest peers run the same deterministic code ($BLAME()$) on the same inputs to do so, they will all exclude the same set of malicious peers.

We have shown that in each failed run, all honest peers exclude the same non-empty set of malicious peers. Eventually, we reach one of two cases. In the first case, the number of unexcluded peers will drop below two; in that case the protocol is allowed to fail and thus there is nothing to show. In the second case, we reach a run in which all peers behave honestly (independently of whether they are controlled by the attacker). 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

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

Fig. 3: Example of 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.

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 [54]. However, if one is willing to depend on a more sophisticated bulletin board with dedicated functionality, the efficiency of DiceMix can be improved. 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-net vectors depending on the DC-net 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 an honest bulletin board, and we can drop the CM (commitment) round. This is secure because the independence of the DC-net vectors is 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 the second run (see Section IV-C2 for an explanation of RV).

2) *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-net vectors. The 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 the 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.

A. Communication

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 peer are dominated by the broadcast of the DC-net array $DC[my][\cdot]$ 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. These communication costs have been shown to be asymptotically optimal for P2P mixing [20].

B. Prototype Implementation

We developed a proof-of-concept implementation of the DiceMix protocol. Our unoptimized 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 [47], [56] to find the roots of the power sum polynomial by the Kaltofen-Shoup algorithm for polynomial factorization [37].

1) *Testbed*: We tested our DiceMix implementation in Emulab [59]. 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

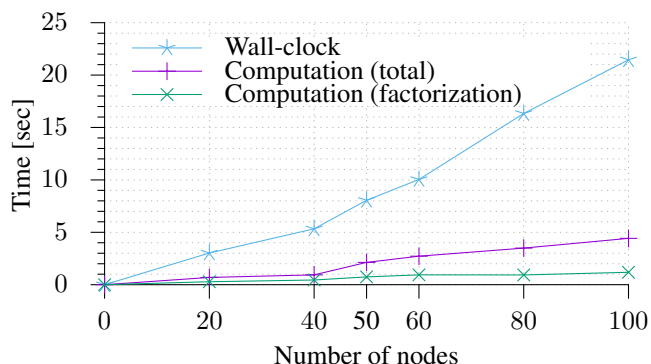


Fig. 4: Wall-clock time and computation times. All peers have a bandwidth of 10 Mbit/s; the bulletin board has a total of 1 Gbit/s; all links have 50 ms latency.

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 20 to 100. 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 with a moderate size of 50 participants, DiceMix runs in about 8 seconds. Second, we measured computation time; the results are depicted in Fig. 4. We considered the average total computation time spent by a peer and average computation time only for polynomial factorization, i.e., solving the equation system involving the power sums.

3) *Optimization*: We observe that solving the equation system is quite expensive, e.g., about one second for 100 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 [34], which provides a highly optimized implementation of the Kaltofen-Shoup algorithm for polynomial factorization over finite fields [37]. Our optimized application solves the equation system involving the power sums in about 0.32 seconds for 100 peers on a 2.70 GHz (Intel Core i7-4800MQ) 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 [52] and Dissent [22] considerably. In comparison, CoinShuffle (as a tailored variant of the Dissent shuffle protocol) needs slightly less than three 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 [5], [6], [40], [46], [51], [53]. Ultimately, cryptocurrencies such as Bitcoin using a public blockchain may in fact provide *less* anonymity than traditional banking, as the deployment of proposed heuristics to the blockchain opens the possibility to know who paid what to whom. Coin mixing has emerged as a technique to overcome this problem while maintaining full compatibility with the current Bitcoin protocol.

A promising solution in this direction is CoinShuffle [52], 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 [43]). However, a run with a decent anonymity set of $n = 50$ peers takes about three minutes to complete [52], *assuming that every peer is honest*. In the presence of f disruptive peers aiming at impeding the protocol, $O(nf)$ communication rounds are required, most of them inevitably taking longer due to the disruptive peers delaying their messages intentionally. For instance, assume that there are $f = 10$

disrupting peers; then 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.

As a solution, we propose CoinShuffle++, a highly efficient coin mixing protocol resulting from the application of DiceMix to the Bitcoin setting.

A. The Bitcoin System

Bitcoin [2], [12], [49] 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 an *address*. Every peer can create new accounts by creating fresh key pairs.

A peer can spend funds associated with her account by creating Bitcoin transactions, which associate funds with another account. In its simplest form, a Bitcoin transaction is composed of a transaction input (a reference to unspent funds in the blockchain associated with some account), a newly created transaction output, and the amount of funds to be transferred from the input to the output. For a transaction to be fully valid, it must be signed with the signing key of the input account.

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

B. Security Goals

Apart from the security goals for a P2P mixing protocol (see Section II-E), a coin mixing protocol must guarantee *correct balance*. It ensures that no funds can be stolen from honest peers.

Correct Balance: For every honest peer p , the total balance of all accounts of peer p is not reduced by running the coin mixing protocol (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 GEN().

Then, for the confirmation subprotocol CONFIRM(), CoinShuffle++ uses CoinJoin [43], [45] 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 transaction thereby constructed transfers the correct amount of funds to their fresh output account. 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 output 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 attack on all other peers who cannot exclude the malicious peer if not convinced of his guilt.

We define CoinShuffle++ in Algorithm 2. There, we denote by $\text{CoinJoinTx}(VK_{in}[], VK_{out}, \beta)$ a CoinJoin transaction that transfers β bitcoins from every input account in $VK_{in}[]$ to the output accounts, where β is a pre-arranged parameter; note that if there are $|P|$ unexcluded peers, then the P2P mixing protocol guarantees that there will be $|M| \leq |P|$ output accounts. Moreover, we denote by $\text{Submit}(tx, \sigma[])$ the submission of tx including all signatures to the Bitcoin network.

1) *Security Analysis:* CoinShuffle++ adheres to the requirements specified in Section IV-B. Thus, sender anonymity and termination in CoinShuffle++ are immediate. (We refer the reader to [45] for a detailed taint-based analysis on the privacy implications of CoinJoin-based coin mixing protocols.) Correct balance is enforced by the CoinJoin paradigm: by construction, a peer signs only transactions that will transfer her 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 a small fee to prevent transaction flooding 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 and thus may need a *change address*. Finally, after honestly performing the CoinShuffle++ protocol, a peer could spend her bitcoins in the input account before the CoinJoin transaction is confirmed, in a double-spending attempt. All these challenges are easy to overcome. We refer the reader to the literature on CoinJoin-based coin mixing, e.g., [43], [45], [52], for details.

DiceMix does not rely on any external anonymous channel

Algorithm 2 CoinShuffle++

```

proc GEN( )
     $(vk, sk) := \text{AccountGen}()$  ▷ Stores  $sk$  in the wallet
    return  $vk$ 

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

```

(e.g., Tor network [23]) for mixing coins. Nevertheless, to ensure unlinkability of inputs of the CoinJoin transaction with network-level details such as IP addresses, using an external anonymous channel is highly recommended both for running DiceMix and actually spending the mixed funds later.

4) *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 makes it possible to define variants of CoinShuffle++ to support a wide range of crypto-currencies and signature algorithms, including interactive signature protocols.

For example, the integration of Schnorr signatures is planned in an upcoming Bitcoin software release [10]. This modification will enable aggregate signatures using an interactive two-round protocol among the peers in a CoinJoin transaction [44]. The first round of this two-round protocol does not depend on the details of the transactions and can be run in parallel to the third round (DC) of CoinShuffle++; this keeps the number of required communication rounds at $4f + 2$.

Given that signatures are often the largest individual part of the transactions, aggregate signatures greatly reduce the size of transactions and thus the transaction fee, thereby making mixing using CoinJoin transactions even cheaper.

5) *Resistance against DoS Attacks by Sybils*: CoinShuffle++ makes sure that disruptive peers in a mixing will be excluded in due course. To avoid that the same peers cannot disrupt further protocol runs either, the bootstrapping mechanism (if executed on the bulletin board) can block the unspent transaction outputs in the blockchain used by the disruptive peers for a predefined period of time, e.g., an hour. (They should not be blocked forever because peers could be unresponsive for legitimate reasons, e.g., unreliable connectivity.)

This ensures that the number of unspent transactions outputs belonging to the attacker limits his ability to disrupt CoinShuffle++ on a particular bulletin board. The attacker can try to overcome the blocking by spending the corresponding funds to create new unspent transaction outputs (*child outputs* of the blocked outputs); however, this is expensive because he needs to pay transactions fees. Moreover, the bootstrapping mechanism can block not only the used transaction outputs but also their child outputs.

VII. RELATED WORK IN CRYPTO-CURRENCIES

We give an overview of the literature on privacy-preserving protocols for crypto-currencies. Related work for P2P mixing protocols is discussed throughout the paper.

A. Tumblers

A *tumbler* provides a backwards-compatible centralized mixing service [11] to unlink users from their funds: several users transfer their funds to the tumbler, which returns them to the users at fresh addresses. The main advantage of a centralized approach is that it scales well to large anonymity sets, because the anonymity set is the set of all users using the service in some predefined time window. However, by using these services naively, a user must fully trust the tumbler: First, anonymity is

restricted towards external observers, i.e., the mixing service itself can still determine the owner of the funds. Second and more important, the users have to transfer their funds to the tumbler, which could just steal them by refusing to return them.

1) *Accountable Tumblers*: Mixcoin [13] mitigates the second problem by holding the tumbler accountable if it steals the funds, but theft is still possible. Blindcoin [57] improves upon Mixcoin in that the tumbler additionally cannot break anonymity.

2) *Blindly Signed Contracts and TumbleBit*: Blindly Signed Contracts [36] and its successor TumbleBit [35] propose an untrusted tumbler based on the combination of blind signatures and smart contracts to solve both aforementioned challenges, i.e., theft and anonymity. To perform ordinary mixing this approach requires at least two transactions to be confirmed sequentially (in two different blocks), whereas CoinShuffle++ requires just one transaction.

TumbleBit supports using the second transaction to send a payment to a recipient directly, which is then on par with CoinShuffle++, which also requires one transaction for mixing and one transaction for sending a payment to a recipient. However, this mode of TumbleBit comes with limitations. First, it requires coordination between the tumbler and the recipient. Second, it requires more fees than CoinShuffle++, because the CoinJoin transaction used in CoinShuffle++ is cheap, in particular if using aggregate signatures. Third, it requires the payment amount to be exactly the mixing amount, which hinders availability severely, because it is very difficult to find enough users that are willing to send the exact same amount of funds at a similar time. With CoinShuffle++, instead, the second transaction, i.e., the actual spending transaction is a normal transaction and supports change addresses, at which peers get their remaining funds back.

B. Other P2P Mixing Approaches

In CoinParty [62], a set of mixing peers is used to mix funds of users. It is assumed that $1/3$ of the mixing parties are honest. This trust assumption is not in line with the philosophy of Bitcoin, which works 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 a fundamental requirement for any protocol providing anonymity.

Xim [8] improves on its related previous work [6] 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 publishing several Bitcoin transactions in the Bitcoin blockchain, which takes on average at least ten minutes for each transaction.

In contrast, CoinShuffle++ requires to submit a single transaction to the Bitcoin blockchain independently on the number of peers.

C. Privacy-preserving Crypto-currencies

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. Nevertheless, several promising designs of crypto-currencies with built-in privacy features are available.

1) *Zerocoin and Zerocash*: Zerocoin [48] and its follow-up work Zerocash [7], whose implementation Zcash has been deployed recently [4], are crypto-currency 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.

2) *CryptoNote*: The CryptoNote design [58] 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 anonymity 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 of the 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 verifying 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 by the Bitcoin Core client [9]. Pruning reduces the storage requirements of nodes drastically by deleting spent transactions from local storage once verified. This is impossible in CryptoNote because it is not entirely clear whether funds in the blockchain have been spent or not. A CoinJoin-based approach such as CoinShuffle++ does not suffer from this problem and is compatible with pruning.

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 protocols.

At the core of the problem is handling of peers that appear to be offline. They cannot be handled like active disruptors: While sacrificing the anonymity of some peer p is not at all a problem if peer p is proven to be an active disruptor and thus malicious, sacrificing the anonymity of p is a serious issue and can render a protocol insecure if p goes 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.

Our attack is based on the well-known and very basic observation that an offline peer cannot possibly have sent a message, which comes in many shapes in basically every anonymous communication protocol with reasonable latency [14], [61]. While the attack relies on this very basic observation, it has been overlooked in the literature that a hard requirement to terminate successfully in the presence of offline peers makes existing P2P mixing protocols vulnerable.

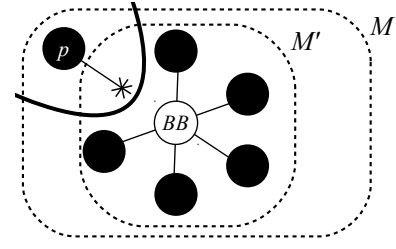


Fig. 5: A P2P Mixing Protocol under Attack. Peer p is partitioned from the bulletin board BB . The dashed rectangles indicate the message sets M and M' of the peers in the respective rectangle.

A. Example: A Deanonymization Attack on Dissent

We illustrate the attack on the Dissent shuffle protocol [22], [55].⁵ 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. 5):

- 1) The network attacker does not interfere with the protocol until the last communication round. In the last round, the attacker partitions the network into 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.⁶ However, anonymity is not broken so far.
- 2) The remaining peers must eventually conclude that peer p is offline and exclude her; otherwise they will not be able to continue the protocol, because they cannot assume that p will ever be reachable again. The strategy by which Dissent provides termination in such a situation is through a wrapper protocol that instructs the remaining peers to attempt a second run of Dissent without peer p . In this second run, the remaining peers resubmit their input messages used in the first run [22, Section 5.4]. The attacker does not interfere with this second run, and so the run will succeed with a final set M' of mixed messages.
- 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 [22], [55] does not

⁵There are several protocols named Dissent. First, there is a P2P mixing protocol proposed by Corrigan-Gibbs and Ford [22] and formally proven secure by Syta et. al. [55]. Second, there is protocol [60] in a client/server setting, which requires trust in one of several servers and is consequently not relevant in our context. The former (P2P) protocol by Corrigan-Gibbs and Ford [22] 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 [22, Section 3].

⁶Dissent has the property that a passive network observer (not participating in the protocol) can also reconstruct M .

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 the wrapper protocol to ensure termination by starting a new run of Dissent without the exposed disruptor whenever a run has failed.

The formal analysis of the Dissent, however, does not cover the wrapper protocol. It considers only a single run of Dissent, and correctly establishes anonymity and accountability for a single run. It has been overlooked that anonymity is lost under sequential composition of several runs of Dissent using the same input messages, as prescribed in the wrapper protocol.

While Corrigan-Gibbs and Ford [22] 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 [22, Section 5.5], their discussion is restricted to reliability and fails to identify the consequences for anonymity.

B. Generalizing the Attack

The underlying reason for this intersection-like attack is a fairness issue: the attacker, possibly controlling some malicious peers, can learn (parts of) the final message set M of a protocol run while denying M to the other peers. If now some peer p appears to be offline, e.g., because the attacker blocks network messages, the remaining peers must finish the protocol without p with a message set M' , which unlike M does not contain m_p . Thus the attacker has learned that m_p belongs to p .

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

Attack Description: We assume an execution of a P2P mixing protocol with peer set $P = \{p_1, \dots, p_n\}$ and their set of fixed input messages $M = \{m_1, \dots, m_n\}$. 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.)

For some i , let r be the first communication round after which input message m_i of peer p_i is known to a collusion of a minority S of peers with $p_i \notin S$.⁷ Such a round exists, because every peer outputs M at the end of a 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. The attacker lets the first $r - 1$ protocol rounds run normally. In round r , he collects the protocol message and

learns m_i (by control of S). Then the attacker selects an index i^* from the set of honest peers. Starting with round r , the attacker only delivers protocol messages not from p_{i^*} and not from his own peers in A ; all these peers appear offline for the remaining peers in $R := P \setminus (\{p_{i^*}\} \cup A)$. By assumption, $|R| \geq 2$, and hence by the termination property, those remaining peers in R will finish the protocol with a public result set $M' \subsetneq M$.

We distinguish cases. If $i^* = i$, then $p_i \notin R$. Since additionally R is a minority, which has not seen any protocol messages from p_i after round $r - 1$, the peers in R do not know m_i , and thus $m_i \notin M'$. If instead $i^* \neq i$, then $p_i \in R$, and the 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 by checking whether $m_i \notin M'$. This breaks the anonymity of p_{i^*} .

C. How DiceMix Avoids the Attack

To avoid the intersection of message sets, DiceMix draws fresh messages in each run. Also, whenever some honest peer p excludes an unreachable honest peer p' (and sacrifices the anonymity of p'), the correct confirmation property will ensure that the current run will not terminate successfully for peer p' , because p' and p will have different views on the current set P of unexcluded peers. Thus no anonymity is required for the current run and malicious and offline peers can be handled equally (as done throughout the previous sections).

IX. CONCLUSIONS

In this work we present DiceMix, a P2P mixing protocol based on DC-nets that enables 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 $4 + 2f$ communication rounds in the presence of f misbehaving peers. We implemented DiceMix and showed its practicality even for a large number of 50 to 100 peers.

We use DiceMix to design CoinShuffle++, a practical decentralized coin mixing protocol for Bitcoin. Our evaluation results show that CoinShuffle++ is a promising approach to ensure unlinkability of Bitcoin transaction while requiring no change to the current Bitcoin protocol.

ACKNOWLEDGMENTS

We thank Bryan Ford for insightful discussions, and the anonymous reviewers for their helpful comments. We also thank Henry Corrigan-Gibbs and Dan Boneh for sharing the manuscript of [20]. This work was supported by the German Ministry for Education and Research (BMBF) through funding for the German Universities Excellence Initiative.

REFERENCES

- [1] "AN.ON (anonymity.online)," <https://anon.inf.tu-dresden.de/>.
- [2] "Bitcoin Developer Guide," <https://bitcoin.org/en/developer-guide>.
- [3] "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/>.

⁷This 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 .

- [4] “Zcash,” <https://z.cash/>.
- [5] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, “Evaluating user privacy in bitcoin,” in *FC’13*.
- [6] S. Barber, X. Boyen, E. Shi, and E. Uzun, “Bitter to better. how to make Bitcoin a better currency,” in *FC’12*.
- [7] 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*.
- [8] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, “Sybil-resistant mixing for Bitcoin,” in *WPES’14*.
- [9] Bitcoin Core, “0.11.0 release notes,” <https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md#block-file-pruning>.
- [10] —, “Segregated witness: the next steps,” <https://bitcoincore.org/en/2016/06/24/segwit-next-steps/#schnorr-signatures>.
- [11] Bitcoin Wiki, “Mixing services,” https://en.bitcoin.it/wiki/Category:Mixing_Services.
- [12] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “SoK: Research perspectives and challenges for bitcoin and cryptocurrencies,” in *S&P’15*.
- [13] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. Kroll, and E. Felten, “Mixcoin: Anonymity for Bitcoin with accountable mixes,” in *FC’14*.
- [14] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, “Denial of service or denial of security?” in *CCS’07*.
- [15] J. Bos and B. den Boer, “Detection of disrupters in the DC protocol,” in *EUROCRYPT’89*.
- [16] D. Cash, E. Kiltz, and V. Shoup, “The twin Diffie-Hellman problem and applications,” *J. Cryptol.*, vol. 22, no. 4, 2009.
- [17] D. Chaum, “The dining cryptographers problem: Unconditional sender and recipient untraceability,” *J. Cryptol.*, vol. 1.
- [18] —, “Untraceable electronic mail, return addresses, and digital pseudonyms,” *Comm. ACM*, vol. 4, no. 2, 1981.
- [19] R. Chien and W. Frazer, “An application of coding theory to document retrieval,” *IEEE Trans. Inf. Theor.*, vol. 12, no. 2, 1966.
- [20] H. Corrigan-Gibbs and D. Boneh, “Bandwidth-optimal DC-nets,” private communication.
- [21] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, “Riposte: An anonymous messaging system handling millions of users,” in *S&P’15*.
- [22] H. Corrigan-Gibbs and B. Ford, “Dissent: Accountable anonymous group messaging,” in *CCS’10*.
- [23] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” in *USENIX Security’04*.
- [24] D. Dolev, R. Reischuk, and H. R. Strong, “Early stopping in byzantine agreement,” *J. ACM*, vol. 37, no. 4, 1990.
- [25] L. A. Dunning and R. Kresman, “Privacy preserving data sharing with anonymous ID assignment,” *IEEE Trans. Inf. Forensic Secur.*, vol. 8, no. 2, 2013.
- [26] M. Florian, J. Walter, and I. Baumgart, “Sybil-resistant pseudonymization and pseudonym change without trusted third parties,” in *WPES’15*.
- [27] C. Franck, “Dining cryptographers with 0.924 verifiable collision resolution,” *Annales UMCS, Informatica*, vol. 14, no. 1, 2014.
- [28] C. Franck and J. van de Graaf, “Dining cryptographers are practical,” arXiv CoRR abs/1402.2269, <https://arxiv.org/abs/1402.2269>.
- [29] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson, “Non-interactive key exchange,” in *PKC’13*, 2013.
- [30] 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.
- [31] D. M. Goldschlag, M. Reed, and P. Syverson, “Hiding Routing Information,” in *Information Hiding: First International Workshop*, 1996.
- [32] P. Golle and A. Juels, “Dining cryptographers revisited,” in *EURO-CRYPT’04*.
- [33] 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>.
- [34] W. Hart, F. Johansson, and S. Pancratz, “FLINT: Fast Library for Number Theory,” 2015, version 2.5.2, <http://flintlib.org>.
- [35] E. Heilman, F. Baldimtsi, L. Alshenibr, A. Scafuro, and S. Goldberg, “TumbleBit: An untrusted tumbler for Bitcoin-compatible anonymous payments,” in *NDSS’17*.
- [36] E. Heilman, F. Baldimtsi, and S. Goldberg, “Blindly signed contracts: Anonymous on-blockchain and off-blockchain Bitcoin transactions,” in *BITCOIN’16*.
- [37] E. Kaltofen and V. Shoup, “Fast polynomial factorization over high algebraic extensions of finite fields,” in *ISSAC’97*.
- [38] W. Kautz and R. Singleton, “Nonrandom binary superimposed codes,” *IEEE Trans. Inf. Theor.*, vol. 10, no. 4, pp. 363–377, 2006.
- [39] S. Kochen, A. Sokolov, and K. Fuller, “IRCV3.2 server-time extension,” 2012, <http://ircv3.net/specs/extensions/server-time-3.2.html>.
- [40] P. Koshy, D. Koshy, and P. McDaniel, “An analysis of anonymity in Bitcoin using P2P network traffic,” in *FC’14*.
- [41] A. Krasnova, M. Neikes, and P. Schwabe, “Footprint scheduling for dining-cryptographer networks,” in *FC’16*.
- [42] D. Krawisz, “Mycelium Shufflepuff (an implementation of CoinShuffle),” <https://github.com/DanielKrawisz/Shufflepuff>.
- [43] G. Maxwell, “CoinJoin: Bitcoin privacy for the real world,” Post on Bitcoin Forum, 2013, <https://bitcointalk.org/index.php?topic=279249>.
- [44] —, “Signature aggregation for improved scalability,” 2016, <https://bitcointalk.org/index.php?topic=1377298.0>.
- [45] S. Meiklejohn and C. Orlandi, “Privacy-enhancing overlays in Bitcoin,” in *BITCOIN’15*.
- [46] 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*.
- [47] A. Mellit, “PARI/GP Python interface,” <https://code.google.com/archive/p/pari-python/>.
- [48] I. Miers, C. Garman, M. Green, and A. D. Rubin, “ZeroCoin: Anonymous distributed e-cash from Bitcoin,” in *S&P’13*.
- [49] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” <https://bitcoin.org/bitcoin.pdf>, 2008.
- [50] V. Y. Pan, “Faster solution of the key equation for decoding BCH error-correcting codes,” in *STOC’97*, 1997, pp. 168–175.
- [51] F. Reid and M. Harrigan, “An analysis of anonymity in the Bitcoin system,” in *SXSW’13*.
- [52] T. Ruffing, P. Moreno-Sanchez, and A. Kate, “CoinShuffle: Practical decentralized coin mixing for Bitcoin,” in *ESORICS’14*, 2014.
- [53] M. Spagnuolo, F. Maggi, and S. Zanero, “Bitlodine: Extracting intelligence from the Bitcoin network,” in *FC’14*, 2014.
- [54] T. K. Srikanth and S. Toueg, “Simulating authenticated broadcasts to derive simple fault-tolerant algorithms,” *Distributed Computing*, vol. 2, no. 2, 1987.
- [55] 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.
- [56] PARI/GP 2.3, The PARI Group, Bordeaux, <http://pari.math.u-bordeaux.fr/>.
- [57] L. Valenta and B. Rowan, “Blindcoin: Blinded, accountable mixes for Bitcoin,” in *BITCOIN’15*.
- [58] N. van Saberhagen, “CryptoNote,” 2013, <https://cryptonote.org/whitepaper.pdf>.
- [59] 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.
- [60] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, “Dissent in numbers: Making strong anonymity scale,” in *OSDI’12*.
- [61] D. I. Wolinsky, E. Syta, and B. Ford, “Hang with your buddies to resist intersection attacks,” in *CCS’13*.
- [62] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, “CoinParty: Secure multi-party mixing of bitcoins,” in *CODASPY’15*.