On the Anonymity Guarantees of Anonymous Proof-of-Stake Protocols

Markulf Kohlweiss[1], Varun Madathil[2], Kartik Nayak[3], and Alessandra Scafuro[2]

[1]University of Edinburgh
[2]North Carolina State University
[3]Duke University

March 26, 2021

Abstract

In proof-of-stake (PoS) blockchains, stakeholders that extend the chain are selected according to the amount of stake they own. In S&P 2019 the "Ouroboros Crypsinous" system of Kerber et al. (and concurrently Ganesh et al. in EUROCRYPT 2019) presented a mechanism that hides the identity of the stakeholder when adding blocks, hence preserving anonymity of stakeholders both during payment and mining in the Ouroboros blockchain. They focus on anonymizing the messages of the blockchain protocol, but suggest that potential identity leaks from the network-layer can be removed as well by employing anonymous broadcast channels.

In this work we show that this intuition is flawed. Even ideal anonymous broadcast channels do not suffice to protect the identity of the stakeholder who proposes a block.

We make the following contributions. First, we show a formal network-attack against Ouroboros Crypsinous, where the adversary can leverage *network delays* to distinguish who is the stakeholder that added a block on the blockchain. Second, we abstract the above attack and show that whenever the adversary has control over the network delay – within the synchrony bound – loss of anonymity is inherent for *any protocol* that provides liveness guarantees. We do so, by first proving that it is impossible to devise a (deterministic) state-machine replication protocol that achieves basic liveness guarantees and better than $(1-2f)$ anonymity at the same time (where f is the fraction of corrupted parties). We then connect this result to the PoS setting by presenting the *tagging* and *reverse tagging* attack that allows an adversary, across several executions of the PoS protocol, to learn the stake of a target node, by simply delaying messages for the target. We show that our attacks are practical, by describing how they can be carried out over the Zcash blockchain network (even when Tor is used). We conclude by suggesting approaches that can mitigate such attacks.

1 Introduction

Lamport, Shostak, and Pease introduced the Byzantine Generals Problem with the goal of achieving consensus among a group of known parties. In their formulation, consensus is achieved even when a fixed fraction of the parties, called Byzantine parties, exhibit arbitrary behavior. Since the parties are *known* to each other at all points of time, these protocols are also referred to as permissioned consensus protocols. With Bitcoin, for the first time, Nakamoto introduced a consensus protocol, or a blockchain, where parties can participate in the protocol without having their identity known to other parties. Such protocols where an identity is not required are called permissionless protocols. In permissionless consensus, an adversary can introduce a large number of parties called Sybils and hence the number of adversaries can always be greater than the fixed fraction allowable by a permissioned consensus

protocol. To address this concern, permissionless protocols rely on the use of constrained *resources*. They achieve security assuming that honest parties hold a majority of the available resources. Two of the most commonly used resources are computation and stake in the system. For instance, Bitcoin [1] uses computation whereas protocols such as Ouroboros [2] [3] and Algorand [4] use stake. These resources are used to elect leaders (or proposers) who are allowed to contribute to the blockchain. The leaders produce a proof of being elected as leaders (e.g., proof-of-work, or a verifiable random function (VRF) output sufficiently close to zero) that can be publicly verified.

Privacy in PoS blockchains. The inherent public nature of permissionless blockchains makes privacy of transactions an important concern – a party can learn about transactions even without participating in the blockchain protocol.

Recently, there have been multiple elegant works that have considered the goal of obtaining privacy in these blockchains. Some of them include ZCash [5], Monero [6], ZEXE [7], based on proof-of-work, and Ouroborous Crypsinous [8], Ganesh et al. [9], and Baldimtsi et al. [10] based on PoS. In a PoW blockchain, at a high-level, privacy of transactions is achieved by encrypting the content of the transactions and providing a zero-knowledge proof that the transaction has been added correctly. In a PoS blockchain, achieving privacy is more involved since the stake of a party is used in two ways: (i) electing leaders/committees who contribute to the extension of the blockchain, and (ii) as a part of transactions that update the stakes of parties. Thus, in order to guarantee the privacy of transactions, it is inherently necessary to ensure that parties' stakes are not revealed based on the execution of the chain extension protocol. On the other hand, for Sybil resistance, the number of times a party is elected is proportional to its stake. This, along with the fact that a public key associated with a VRF is used to provide a verifiable proof of leadership, directly reveals an approximation of the parties' stake. Thus, ensuring privacy of transactions in PoS blockchains while simultaneously providing Sybil resistance is a challenge.

Independently of whether entire transactions are private, keeping the stake private is directly related to the security of the PoS system itself. Otherwise, an adversary can mount a selective attack on a party with a high stake (e.g., via malware or a denial-of-service) either to steal the secret key of the party, or to prevent this party from functioning correctly in the PoS system. In either case, the security of the entire system is weakened since a portion of "honest" stake is prevented from functioning correctly.

Concerns with the approach to privacy. Ouroboros Crypsinous [8] (which we will refer to as Crypsinous) and Ganesh et al. [9], tackle this challenge in a PoS setting. In order to hide the public key associated with a VRF, these works replace VRFs with an *anonymous* verifiable random function (AVRF), where the verification of eligibility is not done using any specific public key, but the set of all public keys in the system. An anonymous VRF guarantees that by seeing the block, no one is able to learn the identity of the party who added the block. This, in addition to the fact that transactions are private, guarantees that by looking at the blockchain protocol messages, no information about the stake of a particular party can be leaked. However, Crypsinous recognizes that protocol messages travel over a public network – modeled as an ideal functionality $\mathcal{F}_{\mathsf{N-MC}}^{\Delta}$ (see Fig.1 [11])– and the adversary can learn information about the identity of an elected party through the leakage of the network channel, e.g., by associating a certain block to a certain IP address. They informally claim that if the underlying communications were carried over an *anonymous* broadcast channel instead, then the network meta-data of the sender is hidden, hence breaking the link between a block and its sender. Since this claim is informal, no particular anonymous channel functionality is provided. The work by [9] instead explicitly describe an ideal anonymous broadcast channel functionality that they introduce, and use that for all the communications. We elaborate further on the meaning of ideal anonymous broadcast channel next.

Ideal anonymous broadcast channels. An ideal anonymous broadcast channel can be described via an ideal functionality $\mathcal{F}_{\mathsf{ABC}}$, that captures the security properties we intuitively can hope to achieve over a network.

A *perfect* definition of an ideal anonymous broadcast (ABC) – $\mathcal{F}_{\mathsf{ABC}}^{\mathsf{perfect}}$ – would be one that takes as input a message m from a party (or adversary) and adds the message m to the buffers of all parties without allowing any influence from the adversary on the delivery of messages. Such a functionality however is very strong, and would trivially imply consensus, since all parties receive the same messages at the same time.

Anonymous Broadcast functionality : $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$

Any party can register (or deregister). Let the list of registered parties be $\mathbb{P} = \{P_1 \dots P_n\}$. The functionality maintains a message buffer M.

- *Honest sender send*: Upon receiving $(\texttt{SEND}, \mathsf{sid}, m)$ from some party $P_s \in \mathbb{P}$, where $\mathbb{P} = \{P_1, \dots, P_n\}$ denotes the current party set, do :
  1. Choose n new unique message-IDs $\mathsf{mid}_1, \dots, \mathsf{mid}_n$
  2. Initialize $2n$ new variables $D_{\mathsf{mid}_1} := D_{\mathsf{mid}_1}^{MAX} \dots D_{\mathsf{mid}_n} := D_{\mathsf{mid}_n}^{MAX} := 1$. These are the delays and the maximum delays of the message for each party.
  3. Set $M = M||(m, \mathsf{mid}_i, D_{\mathsf{mid}_i}, P_i)$ for each $P_i \in \mathbb{P}$
  4. Send $(\texttt{SEND}, m, \mathsf{mid}_1, \dots, \mathsf{mid}_n)$ to the adversary.
- *Honest party fetching* : Upon receiving $(\texttt{FETCH}, \mathsf{sid})$ from $P_i \in \mathbb{P}$:
  1. For all tuples $(m, \mathsf{mid}, D_{\mathsf{mid}}, P_i) \in M$ set $D_{\mathsf{mid}} := D_{\mathsf{mid}} - 1$.
  2. Let $M_0^{P_i}$ denote the subvector of M including all tuples of the form $(m, \mathsf{mid}, D_{\mathsf{mid}}, P_i)$ with $D_{\mathsf{mid}} = 0$ (in the same order as they appear in M). Delete all entries in $M_0^{P_i}$ from M and send $M_0^{P_i}$ to P_i.
- *Adding adversarial delays* : Upon receiving $(\texttt{DELAY}, \mathsf{sid}, (T_{\mathsf{mid}_1}, \mathsf{mid}_1), \dots, (T_{\mathsf{mid}_\ell}, \mathsf{mid}_\ell))$ from the adversary, do the following for each pair $(T_{\mathsf{mid}_i}, \mathsf{mid}_i)$:
  1. If $D_{\mathsf{mid}_i}^{MAX} + T_{\mathsf{mid}_i} \leq \Delta$ and mid is a message-ID registered in the current M, set $D_{\mathsf{mid}_i} := D_{\mathsf{mid}_i} + T_{\mathsf{mid}_i}$ and set $D_{\mathsf{mid}_i}^{MAX} := D_{\mathsf{mid}_i}^{MAX} + T_{\mathsf{mid}_i}$; otherwise ignore this pair.
- *Adversarial sender (partial) multicast*: Upon receiving $(\texttt{MSEND}, (m_1, P_1), \dots, (m_\ell, P_\ell))$ from the adversary with $P_1, \dots, P_\ell \in \mathbb{P}$:
  1. Choose ℓ new unique message-IDs $\mathsf{mid}_1, \dots, \mathsf{mid}_\ell$
  2. Initialize 2ℓ new variables $D_{\mathsf{mid}_1} := D_{\mathsf{mid}_1}^{MAX} \dots D_{\mathsf{mid}_\ell} := D_{\mathsf{mid}_\ell}^{MAX} := 1$
  3. Set $M = M||(m_1, \mathsf{mid}_1, D_{\mathsf{mid}_1}, P_1)|| \dots ||$ $(m_\ell, \mathsf{mid}_\ell, D_{\mathsf{mid}_\ell}, P_\ell)$
  4. Send $(\texttt{MSEND}, m_1, \mathsf{mid}_1, \dots, m_\ell, \mathsf{mid}_\ell)$ to the adversary.

Figure 1: $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$

A more realistic candidate for an ideal anonymous broadcast functionality, would be one that follows the definition of the original broadcast functionality $\mathcal{F}_{\mathsf{NMC}}^{\Delta}$ [11] (for which there exist candidate protocols) and slightly modify it so that the identity of the sender of a message m is not revealed to the adversary. In such a formalization, the adversary can still influence the buffer of honest parties, by

introducing targeted delays (within the synchronous bound Δ) to the messages sent by both honest and corrupt senders. This is the formulation of ideal anonymous broadcast channel that we use in our paper (described in Fig. 1) and we denote by $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$.

Ganesh at al. [9] introduce a different formulation of anonymous broadcast where an adversary is only allowed to delay messages sent by malicious parties (cf. [9], page 10). Nevertheless, the adversary is still allowed to add messages to the buffers of targeted honest parties. Looking ahead, this adversarial capability is sufficient to mount the attacks we propose in this paper. This is because the crux of our attack is to ensure that honest parties have different views, which is directly allowed in their definition since the adversary can influence the buffer of honest parties.

Our Contribution

In this work, we show that while the claim of using an anonymous broadcast channel seems reasonable at first, this does not suffice to hide the stakes of the parties. Specifically, we have the following contributions:

- **An anonymity attack in Crypsinous* that leverages network delay.** We show that even in the presence of *ideal anonymous* broadcast channels, an adversary is able to leverage a synchronous network delay to partition the views of the parties so that, when a block is published, it reflects the view of the block proposer, hence directly linking the block to the network identity of its proposer. Formally, we show that, in contrast with the informal claims of Crypsinous [8], they cannot securely instantiate the ideal private-ledger functionality *without leakage*. We do so, by showing an environment that always distinguishes the simulated transcript from the real world transcript of Crypsinous* (where * here denotes the version of Crypsinous augmented with ideal anonymous broadcast channels) with a non-negligible probability. A similar attack can be applied to the protocol of Ganesh et al. [9], even in presence of their own ideal anonymous broadcast functionality, since the adversary only needs to influence the buffer of the honest parties which they allow.

- **Impossibility of anonymous (deterministic) PoS blockchain protocols.** We show that when parties receive inputs at different times (due to network delays), it is impossible to devise a PoS blockchain protocol where both liveness and anonymity are guaranteed. We do so in two steps. First, we show a lower bound for the existence of one-shot deterministic state-machine replication protocol that achieves both (z, t)-liveness and $(1-2f)$-anonymity, where (z, t)-liveness means that if a transaction was received by a z fraction of honest parties more than t time ago, it should be added in the block, and $(1 - 2f)$-anonymity means that every message sent by an honest party in the protocol is anonymous within an $(1-2f)$ fraction of the parties participating in the protocol. Then we map this lower bound to the PoS setting by presenting the *tagging* and *reverse tagging* attacks. These allow an adversary, across several execution of a PoS protocol that satisfies liveness, to learn the stake of a set of target nodes (or a specific node), by simply delaying messages for the target and corrupting f stakeholders.

- **Practicality of the attacks.** Our attacks rely on the capability of the adversary to control the network delay incurred by the targeted parties, in order to influence their local view. In practice, messages may reach their recipient quickly and it is unclear how an adversary can prevent a party from receiving a message from other peers.

  Nonetheless, we show that this type of attack can be carried out on the Zcash blockchain even by a low-resource adversary.

  We chose Zcash due to its similarity with Crypsinous and because, so far, there exist no implementations of privacy-preserving PoS blockchains.

In the following subsections, we elaborate on the intuition for our attacks.

1.1 An Anonymity Attack to Crypsinous*

Before describing the intuition behind the attack, we make two observations for private PoS protocols. First, despite the privacy guarantees, in any protocol, the sender of a transaction will need to know whether the transaction has been committed. This is necessary for the functioning of any blockchain protocol. This allows the adversary to obtain a mapping between a transaction it created and the block in which it was added. Second, since these blockchain protocols assume synchrony in the network, any message received by an honest party will be received by all honest parties within Δ time, where Δ is a pessimistic bounded network delay. However, within the Δ time delay, an adversary can choose an arbitrary delay for its arrival time. Note that such an adversarial capability respects the threat model, since the definition of anonymous broadcast (Fig. 1) allows this. Specifically, it can make a transaction take 0 time for some party while it takes Δ time for another party. Using these two properties, we show that, even if all communications use ideal broadcast channels (that hide the identity of a sender), the adversary can still de-anonymize messages sent by specific parties.

Specifically, the adversary can perform an attack where it is trying to judge the frequency with which a party P proposes a block compared to the rest of the network; this is directly related to P's stake. In PoS protocols such as Crypsinous, the eligible block proposers propose blocks at discrete intervals of time. Suppose these intervals are denoted as $t, t+1, t+2, \ldots$. The adversary sends a unique transaction `txn` to the party P at time $< t$. Of course, party P will share this transaction with the rest of the network, but due to the network delay assumption, they will not receive it before time $t+\Delta$. Whenever a party is elected, if it includes all the transactions in its own view in the next block, then the adversary can perform a simple test to check whether P was elected: if `txn` was included in the block, then P was elected, otherwise, it was not. We stress that merely encrypting transactions does not help since a party should always be able to learn whether its own transaction was included in the chain. Also, the adversary succeeds only probabilistically. However, by repeating the attack multiple times, it can learn the approximate stake of party P.

In Section 3 we present a formal attack against the UC-security of Crypsinous*. We show an environment that is able to distinguish the protocols transcript from the transcript generated by a simulator interacting with an ideal functionality that does not leak the identity of the block proposer.

1.2 Impossibility of Anonymous (Deterministic) PoS Protocols

We leverage the idea behind the above attack and show that when the adversary can cause parties to have different local views there exists no PoS protocol that can guarantee anonymity of stakeholders, if liveness must be guaranteed.

To prove this, we consider the notion of anonymous one-shot state-machine replication protocol with (z,t)-liveness and a-anonymity for parameters z, t and f. Here, (z,t)-liveness means that if a z fraction of honest parties have received a transaction at time $\leq t$, this transaction should be given in the output; a-anonymity means that every message sent by an honest party in the protocol is anonymous within an a-fraction of the parties participating in the protocol. We then prove that it is impossible to construct a deterministic protocol that guarantees both (z,t)-liveness and better than $(1-2f)$-anonymity. The main idea of the lower bound is to show that *any gap* (e.g., delay) between the times with which inputs arrive at the parties, must manifest in the output of the protocol – if liveness has to be provided. The generalization with parameter t shows that simply waiting for transaction to be old enough does not suffice to obtain anonymity. We consider an attack where n participants are partitioned in three sets P, Q and R, where P and Q are of size f fraction of n while R is of size $(1-2f)$ fraction of n. We then construct a sequence of worlds; in each world an input v is received by set R, Q and P with different time configurations (e.g., R receives it first, R and Q receive it first, etc). We show that, if the state-machine replication protocol outputs v in any of the worlds, then the participation with input-dependent messages of a specific set of parties will be exposed, which should not happen if $(1-2f)$-anonymity must be guaranteed. Hence, anonymity demands that the protocol does not output v in any of the worlds. However, such a protocol, while satisfying anonymity would

not satisfy liveness, since there exists a world where enough parties received v early enough, in which case the protocol should output v. We present this ideas formally in Section 4.

We then show the implication of the strategy we used for our lower bound to violate anonymity in the PoS setting. The key observation is that in the PoS setting, participation of a party in the protocol is related to the amount of stake the party owns. We show that an adversary, who has no information about the stake distribution, by simply partitioning the network and offering different views to the parties, is able to identify sets P, Q and R and consequently learn that they hold respectively f, f and $1 - 2f$ fractions of the total stake. We call this the *tagging attack*, which can be mounted even without corrupting any protocol player, and present it in Section 5.1. Furthermore, we present a *reverse tagging attack* where an adversary can target a specific network party and learns the stake it owns. This attack is shown in Section 5.2. Both attacks work for any deterministic PoS protocol that is secure against malicious parties making up f fraction of the total stake.

1.3 Practicality of our attacks

Our attacks leverage the ability of an adversary to delay messages only for a set of targeted parties, up to the synchrony bound Δ. This is a worst-case power given to the adversary when analyzing the robustness of a distributed protocol. In real networks, however, messages might travel much faster than Δ, so one might wonder whether a realistic adversary – who does not control the network globally – is able to prevent a party from receiving a message before Δ, and thus mount our attacks in practice.

Contrary to this intuition, we show that, even on a fast network, delays can be induced by an adversary by leveraging communication-related implementation details of blockchain software. Since there are no implementations of privacy-preserving PoS blockchains, we present our attacks on a privacy-preserving PoW blockchain: Zcash [12]. We describe how an adversary can delay the delivery of a transaction to a targeted Zcash node, by exploiting implementation-specific behavior in the Zcash software. Specifically we leverage the *Invblocking* procedure presented by Miller et al. [13]. This attack exploits an optimization used to advertise new transactions in the Zcash peer-to-peer network, that we describe in Section 6. We give a high-level overview here. When a Zcash node, say Z, learns a new transaction tx, it will first send a short digest H_{tx} to its peers before sending tx. Each peer first checks whether they have already seen H_{tx}. If a peer, say A, has not received H_{tx} before, it replies with a `GETDATA` request to obtain the full transaction. At this point, A will wait to hear from Z (and not from any other peer) the full transaction, until a timeout (2 min).[1] An adversary can exploit this implementation detail to delay A. It just needs to corrupt one Zcash node and behave like Z.

We notice that this attack is still possible even if A connects over anonymous channels (e.g., it uses Tor [14] or [15]), since Z can still establish a connection with A and then mount the *Invblocking* procedure described above. In fact, as shown by Biryukov et al. [16], we observe that when Zcash nodes use Tor, they are even more susceptible to delay attacks, since an adversary can leverage implementation peculiarities of both Zcash and Tor. We discuss this in details in Section 6.

1.4 Mitigations

Finally, we propose ideas to mitigate the above attacks. We note that our attacks crucially leverage the ability to present different inputs to the parties by controlling network delays, and the fact that the output of the protocol reflects the inputs of the participants. What if we were able to ensure that parties do not speak based on their local view, but on a view of the network as a whole? We introduce the concept of view *sanitization*, referring to a process by which a party sanitizes its local view by sampling a view computed collectively by the network. The sanitization process would require honest parties distributed across the network to collaborate in crafting the sanitized view. We call these parties sanitizers. As expected, identifying honest parties that can serve as sanitizers is a challenge. We elaborate on these challenges in Section 7.

[1] See Line 2171 of https://github.com/zcash/zcash/blob/master/src/net.cpp

2 Related Work

Many attacks have been proposed in literature and they can be broadly classified as de-anonymization attacks that do clustering of pseudonyms of parties to link transactions to the actual party [17], [18], [19] and de-anonymization attacks that analyze the network [20] [21] [22] [23] [24] [25] that leverage the network to de-anonymize parties in cryptocurrencies. The attack of [23] leverages the unfairness in anonymous communication protocols. In unfair protocols the adversary is allowed to peek at the output when all other parties observe that the protocol failed. This leads to an *intersection attack*. As we shall see later, this is somewhat similar to our attack where the unfairness is the delay an adversary is allowed to set for parties. In [26] the authors describe how a denial of service attack lowers anonymity as messages need to get retransmitted to be delivered, presenting more opportunities for attack for certain anonymous communication systems.

In [24], (Danaan-gift Attack) the adversary donates a small tainted amounts of Zcash to the target's shielded address in hope that the tainted value would remain when the value is de-shielded. Our work differs from other works since we de-anonymize the identity of parties on PoS even if one assumes anonymous channels. We note that network related attacks in privacy preserving blockchains have been acknowledged in previous works such as Zerocash [5] (See Section VI-C), where they discuss a "poison-pill" block to target a user and Quis-Quis [27] where they assume network analysis attacks are out of scope of their paper.

In [13] the authors introduce a "decloaking" method to find influential nodes in the Bitcoin network that are well connected to a mining pool. This attack is similar to our attacks in the sense that they find nodes on the Bitcoin network with highest computational power, where as we find nodes that have higher stakes. In their setting the nodes are not anonymous and communication is done over a public network, where as we assume anonymous stakeholders as well as ideal anonymous channels. Yet, their approach can be effectively used in our scenario as well.

In [28], the authors present an anonymity trilemma. They analyze the relationship between bandwidth overhead, latency overhead, and sender anonymity or recipient anonymity against a global passive (network-level) adversary. They show that any anonymous communication protocol can only achieve two out of the following three properties: strong anonymity, low bandwidth overhead, and low latency overhead. In contrast, in our work we focus on anonymous PoS protocols and prove that there is a tension between liveness and anonymity, and present a lower bound on the anonymity one can hope to achieve in anonymous PoS protocols.

3 An Anonymity Attack to Crypsinous*

In this section, we describe the anonymity attack that can be mounted on Crypsinous*. Specifically, first, we describe the leakage on the proposer identity allowed by the original Crypsinous [8]'s protocol. We then explain how [8] conjectures that this leakage can be avoided if the protocol messages of Crypsinous are exchanged over *ideal anonymous* broadcast channels. We call this version Crypsinous*. Finally, we show an anonymity attack on Crypsinous* that refutes this claim. In other words, we show Crypsinous and anonymous broadcast do not compose in the way envisioned by the authors.

Crypsinous: Privacy for transactions only. Crypsinous provides privacy guarantees to transactions by having a block contain only encryptions of the transactions along with a zero-knowledge (zk) proof stating that the block was formed correctly by an eligible stakeholder. Thus, by just observing the blocks on the blockchain, an adversary only learns that "a party P has computed a private transaction" without learning who P is or what the transaction is about.

However, an adversary observing network packets may gain additional information – for instance, the adversary might learn the network identity of the block proposer for a given block or the identity of the sender of a transaction. Crypsinous acknowledges that their protocol will incur such a leakage when not using communication channels that provide anonymity.

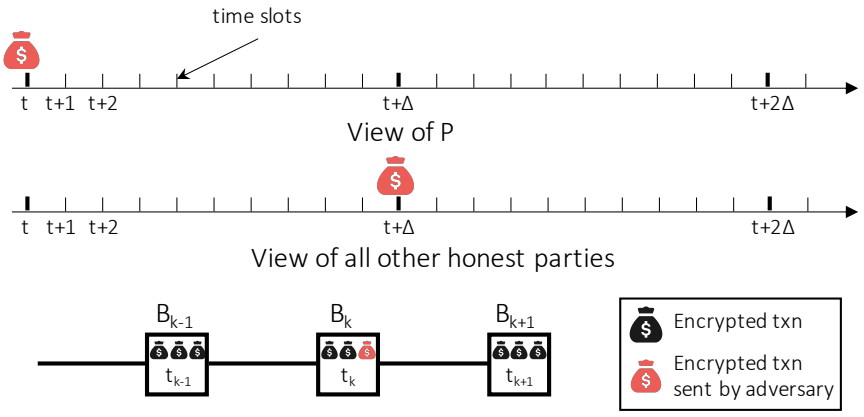

Figure 2: Overview of the attack

To model the above process, they introduce an ideal private ledger functionality $\mathcal{G}_{\mathsf{PL}}$ (see Fig. 1 in Page 7 of [8]). Privacy of transactions is captured by the functionality by only revealing a *blinded* version of the transaction's contents that were added to the chain along with the sender of the transaction. To account for what an adversary can learn about block proposers through the network channels, the ideal functionality provides a *leakage* to the adversary (denoted by $\mathsf{Lkg}_{\mathsf{lead}}$ in [8]). This informs the adversary about the parties that would be selected according to the eligibility function.

Anonymous broadcast channels provide privacy to proposers. Do they? Crypsinous states that: "*If we assumed anonymous broadcast communication channels, the submitter (i.e., the block proposer) would not be needed to be leaked, i.e., the requirement of leaking the submitter is strictly due to network leakage.*" (see footnote at Page 7 of [8]). Thus, if we had an ideal channel to anonymously broadcast messages, then it is not possible to link a message to any specific honest party.

We now present an explicit attack even in the presence of anonymous broadcast channels. An overview is presented in Figure 2. The horizontal scale represents time and the different time ticks represent slots at which "leader elections" are held, i.e., parties attempt to propose the next block. If a party succeeds at time slot t_k in winning the lottery and to mine a block B_k, then it adds all the transaction in its view and adds it to the block. An example blockchain is represented at the bottom. All the transactions are encrypted; however, an adversary can identify transactions produced by itself. Thus, while all encrypted transactions are represented in black, an adversarial transaction is represented in red. In the example, the adversary sends the red transaction at time t to a party P but due to a network delay (which may be imposed by the adversary), all other parties receive the transaction only at time $t+\Delta$. Thus, if party P is elected at time $t \leq t_k < t+\Delta$, then it adds the red transaction in block B_k. Note that we still respect the threat model of Crypsinous and do not assume that the block interval is less than the network delay. For our attack to succeed we only require that a block be produced by P between time t and $t+\Delta$. The adversary can identify that the lottery winner was P since (i) it can identify its own transaction in B_k, and (ii) the time slot at which the block was mined is available in the clear. For any other party, if they are elected in this time frame, B_k will not contain the red transaction. The attacker uses this information to distinguish between P and other parties. In the next sections we formalize this attack, by showing an adversary that can violate the UC-security guarantees of Crypsinous*.

3.1 Universal Composability in Brief

Before we formally present the attack we present a quick primer on universal composability (UC) [29]. In UC security we consider the execution of the protocol in a special setting involving an environment

machine $\mathcal{Z}$, in addition to the honest parties and adversary. In UC, ideal and real models are considered where a trusted party carries out the computation in the ideal model while the actual protocol runs in the real model. The trusted party is also called the ideal functionality. For example the ideal functionality $\mathcal{G}_{\mathsf{PL}}$ of [8] is a trusted party maintaining an ideal private ledger. In the UC setting, there is a global environment (the distinguisher) that chooses the inputs for the honest parties, and interacts with an adversary who is the party that participates in the protocol on behalf of dishonest parties. At the end of the protocol execution, the environment receives the output of the honest parties as well as the output of the adversary which one can assume to contain the entire transcript of the protocol. When the environment activates the honest parties and the adversary, it does not know whether the parties and the adversary are running the real protocol –they are in the real world, or they are simply interacting with the trusted ideal functionality, in which case the adversary is not interacting with any honest party, but is simply "simulating" to engage in the protocol. In the ideal world the adversary is therefore called simulator, that we denote by $\mathcal{S}$.

In the UC-setting, we say that a protocol securely realizes an ideal functionality, if there exist no environment that can distinguish whether the output he received comes from a real execution of the protocol between the honest parties and a real adversary $\mathcal{A}$, or from a simulated execution of the protocol produced by the simulator, where the honest parties only forward date to and from the ideal functionality.

The transcript of the ideal world execution is denoted $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)$ and the transcript of the real world execution is denoted $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$. A protocol is secure if the ideal world transcript and the real world transcripts are indistinguishable. That is, $\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \equiv \{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$.

3.2 Formal Attack to UC-security

According to the definition of UC-security as shown in the previous section a protocol is secure if an environment is unable to distinguish between a real world execution and an ideal world execution. Therefore in Crypsinous*, to formally show an attack, we show an environment $\mathcal{Z}$ that causes every simulator $\mathcal{S}$ interacting with $\mathcal{G}^*_{\mathsf{PL}}$ (i.e., $\mathcal{G}_{\mathsf{PL}}$ without leakage) in the ideal world, to generate a transcript (denoted by $\mathsf{IDEAL}_{\mathcal{G}^*_{\mathsf{PL}},\mathcal{S},\mathcal{Z}}$) that is distinguishable from the transcript the real world adversary $\mathcal{A}$ generates when executing the actual protocol Crypsinous* in the real world where it still has access to the ideal functionalities $\mathcal{F}^{\Delta}_{\mathsf{ABC}}$ (an anonymous broadcast functionality, presented in Fig 1) and $\mathcal{F}_{\mathsf{NIZK}}$ (a zero knowledge functionality). We denote this real-world execution as $\mathsf{REAL}_{\text{Crypsinous*},\mathcal{A},\mathcal{Z}}$. Towards showing this, we should first explain the ideal functionalities $\mathcal{G}^*_{\mathsf{PL}}$ and $\mathcal{F}^{\Delta}_{\mathsf{ABC}}, \mathcal{F}_{\mathsf{NIZK}}$, describe the entire protocol Crypsinous* and then show how the simulator $\mathcal{S}$ interacts with $\mathcal{G}_{\mathsf{PL}}$ and how $\mathcal{A}$ interacts with the protocol and $\mathcal{F}^{\Delta}_{\mathsf{ABC}}$, $\mathcal{F}_{\mathsf{NIZK}}$. In the following, we will report relevant parts of $\mathcal{G}^*_{\mathsf{PL}}$ and the protocol which are sufficient to provide a gist of the attack. The reader can consult [8] for the full specification.

Abridged Version of the Ideal functionality $\mathcal{G}^*_{\mathsf{PL}}$ without leakage The ideal functionality $\mathcal{G}^*_{\mathsf{PL}}$ is shown in Figure 3. In the actual $\mathcal{G}_{\mathsf{PL}}$ functionality, the simulator could read the honest inputs to the functionality, denoted - $\mathcal{I}^T_{\mathcal{H}}$ and know which party was activated with the `MAINTAIN-LEDGER` command. In $\mathcal{G}^*_{\mathsf{PL}}$ we hide this information. But we cannot simply omit this information from $\mathcal{I}^T_{\mathcal{H}}$, then the simulator would have no idea if this was a party that would win the leader election in the real world or not. And thus the simulation is trivially impossible. To this end, we have the $\mathcal{G}^*_{\mathsf{PL}}$ functionality add a bit b that indicates if a party would win, along with the honest input command. The bit b is set if the party is eligible to participate according to the selection function (denoted ϕ). Thus the simulator does not learn the identity of the party, but does learn if the party that sent this input would win in the real world or not.

Abridged ledger functionality with no leakage $\mathcal{G}^*_{\mathsf{PL}}$

The functionality manages a fixed ledger state state, a buffer of unconfirmed transactions buffer, a sequence of honest input $\mathcal{I}^T_{\mathcal{H}}$, and a ptr_P for each party P indicating its local state, i.e., the length of the prefix of state which is visible to P. $\vec{\mathsf{ptr}}$ is used to refer to a vector of all parties' local state pointers. α_P is the relative stake of the party P and ϕ_f is the slot leader assignment function.

Honest party interaction

Upon receiving an input I from a party P at time τ

1. (Extend chain) If I is a `MAINTAIN-LEDGER` command, determine stake of P and toss a coin b that outputs 1 with probability $\phi_f(\alpha_P)$. Update $\mathcal{I}^T_{\mathcal{H}} = \mathcal{I}^T_{\mathcal{H}} \| (I, b, \tau)$
2. (Add transaction) If I is a `SUBMIT` transaction tx, add tx to the buffer. The blinded version of tx hides the recipient and the amount, but the sender of the transaction S is revealed to $\mathcal{A}$. $\mathcal{I}^T_{\mathcal{H}}$ is updated as $\mathcal{I}^T_{\mathcal{H}} = \mathcal{I}^T_{\mathcal{H}} \| (I, \mathit{blind}(\mathsf{tx}), \tau)$ and send $\mathit{blind}(\mathsf{tx})$ to $\mathcal{A}$
3. (Read chain) If I is a `READ` command then it receives a blinded version of the state, tailored for P, that is state_P. (The adversary can set this state). The blinded version ensures that P can only use transactions that are meant for itself.

Adversarial interaction

1. (Read) Upon receiving a `READ` command from the adversary send back the blinded transactions and a blinded version of $\mathcal{I}^T_{\mathcal{H}}$.
2. (Delay) The adversary sends $(\texttt{SETSLACK}, \vec{\mathsf{ptr}})$ and the ledger updates the pointer for each P.
3. The adversary proposes a new block `NEXT-BLOCK`. $\mathcal{G}_{\mathsf{PL}}$ updates the state with this block if the block is valid, else it updates the state with a default block.

Figure 3: Informal description of the modified ledger functionality of [8].

Relevant Protocols from Crypsinous We focus only on the implementation of `MAINTAIN-LEDGER` command that allows a party P to extend the ledger.

`MAINTAIN-LEDGER` is executed as in Protocol $\Pi_{\texttt{MAINTAIN-LEDGER}}$ in Figure 4. We describe the protocol in a $\mathcal{F}^{\Delta}_{\mathsf{ABC}}$ hybrid world (see Fig 1), i.e. assuming an anonymous network is available. At a high-level, each party retrieves the latest chains and transactions from the network, by sending `FETCH` to the functionality $\mathcal{F}^{\Delta}_{\mathsf{ABC}}$. This is its local view. It then checks if is is eligible to extend the chain, and if so, picks the longest local valid chain (see [2]) and extends it with its block.

Abridged Protocol $\Pi_{\texttt{MAINTAIN-LEDGER}}$

We represent a chain using $\mathcal{C}$. To each user is associated a buffer and a $\mathcal{C}_{loc}$. st is used to denote the transaction part of a block.

1. Execute $\mathsf{FetchInformation}$ - Send $\texttt{FETCH}$ to $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$ and receive chains $\mathcal{C}_1 \dots \mathcal{C}_M$ and transactions $(\mathsf{tx}_1 \dots \mathsf{tx}_k)$. Pick the longest valid chain $\mathcal{C}_{loc}$.

   Add $(\mathsf{tx}_1 \dots \mathsf{tx}_k)$ to buffer.

2. Run $\mathsf{StakingProcedure}(\mathsf{buffer}, \mathcal{C}_{loc})$ as follows:

   Check if eligible, by evaluating $\phi_f(\alpha_P)$ and an AVRF. If eligible:

   (a) For each $\mathsf{tx} \in \mathsf{buffer}$, add the valid tx to st.

   (b) Create a zero knowledge proof π proving that the party is eligible.

   (c) Create a block $\mathsf{B} \leftarrow (\pi, \mathsf{st})$ and extend local chain $\mathcal{C}_{loc} \leftarrow \mathcal{C}_{loc} \| \mathsf{B}$

   (d) Send $(\texttt{SEND}, \mathcal{C}_{loc})$ to $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$

Figure 4: Ledger Maintenance Protocol

Attack: no UC-simulator exists if $\mathcal{G}_{\mathsf{PL}}^{*}$ does not leak the identity of the block proposer. We now show an environment that is able to set up the inputs to the parties, and network delay, that can easily distinguish the ideal world from the real world.

The environment $\mathcal{Z}$ creates three parties P_1, P_2 and P_3 and registers them to the system, with a certain stake distribution say D. $\mathcal{Z}$ also initializes the buffer for each of the parties as $\mathsf{buffer}_1 = \mathsf{buffer}_2 = \mathsf{buffer}_3 = \phi$. Once the parties are initialized, $\mathcal{Z}$ proceeds as in Fig 5

The real world execution $\mathsf{REAL}_{\mathcal{A},\textbf{Crypsinous*}}$ In the real world executions, command $\texttt{MAINTAIN-LEDGER}$ is instantiated with protocol $\Pi_{\texttt{MAINTAIN-LEDGER}}$. As shown in Figure 4, in this protocol, each party P_i would first fetch the most updated view from the network by sending $\texttt{FETCH}$ to $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$ and update their buffer_i accordingly. Then, it checks if it is eligible, and if so extends the chain with transactions in buffer_i. Assume that $\mathcal{Z}$ activates the players with stake distribution $\mathsf{D}_1 = (P_1, 0.001\$)$, $(P_2, 0.001\$)$, $(P_3, 1\$))$. In this case, P_3 is the only party who might be eligible to extend the chain. Since the delay for parties P_1 and P_2 is set as immediate delivery $((P_1, 0), (P_2, 0), (P_3, \Delta))$, both parties will have tx^* in their buffer, whereas P_3 will not. P_3 does not include tx^* in $\mathcal{C}^*$ hence $\mathcal{Z}$ will output 1 with probability equal to the probability of P_3 winning the leader election. We assume an eligible party would proceed with computing a block right away. Now because of the stake distribution we can say $Pr[P_3 = \mathsf{leader}] \approx 1$. Hence, $Pr[\mathcal{Z} \rightarrow 1 | \mathsf{REAL}_{\mathcal{A},\mathrm{Crypsinous}^*}] = Pr[P_3 = \mathsf{leader}] \approx 1$.

Environment $\mathcal{Z}$

1. Activate the players P_1, P_2, P_3 with stake distribution $\mathsf{D}_1 = (P_1, 0.001\$)$, $(P_2, 0.001\$)$, $(P_3, 1\$)$.
2. Let parties P_1, P_2, P_3 make transactions for tiny transfers (e.g., 0.000001 units) in order to populate the chain without altering their stake.
3. For each slot let all parties query `MAINTAIN-LEDGER` and populate the blockchain accordingly. For each `SEND`, sent to $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$, the transaction is added to the buffers of P_1, P_2 and P_3.
4. At a certain slot sl^*, let the buffers of the parties be $\mathsf{buffer}_1 = \mathsf{buffer}_2 = \mathsf{buffer}_3 = \mathsf{buffer}$. At this point, instruct P_1 to make a transaction tx^* addressed to P_2. P_1 sends $(\mathtt{SEND}, \mathsf{tx}^*)$ to $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$. Instruct the adversary $\mathcal{A}$ to set the following vector of delays to $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$: $((P_1, 0), (P_2, 0), (P_3, \Delta))$
5. In slot $\mathsf{sl}^* + \Delta$, all parties are instructed to perform `MAINTAIN-LEDGER`. Let $\mathcal{C}^*$ be the chain sent to $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$.
6. Decision: If $\mathcal{C}^*$ does not include tx^*, output 1. Else output 0.

Figure 5: The distinguishing environment

The ideal world execution $\mathsf{IDEAL}_{\mathcal{S},\mathcal{G}_{\mathsf{PL}}^*}$ Now, consider the ideal world.

In the ideal world $\mathcal{Z}$ instructs each P_i to invoke `MAINTAIN-LEDGER`. Each P_i sends $I =$ `MAINTAIN-LEDGER` to $\mathcal{G}_{\mathsf{PL}}^*$. $\mathcal{G}_{\mathsf{PL}}^*$ updates $\mathcal{I}_{\mathcal{H}}^T$ and sends I to the simulator $\mathcal{S}$. The simulator will prepare the block by following protocol $\Pi_{\mathtt{MAINTAIN\text{-}LEDGER}}$ but leveraging the fact that it can simulate the behaviour of $\mathcal{F}_{\mathsf{NIZK}}$ and $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$. By simulating the network functionality $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$ the simulator is able to learn the transactions that are sent over the network by the real world parties. Moreover $\mathcal{S}$ learns the delays that the environment $\mathcal{Z}$ instructed the real world adversary to set.

At slot sl^*, the simulator learns that a party has been elected and attempts to run $\Pi_{\mathtt{MAINTAIN\text{-}LEDGER}}$. The first step is to retrieve the information from $\mathcal{F}_{\mathsf{ABC}}^{\Delta}$. In this case, $\mathcal{S}$ learns the delay configuration $((P_1, 0), (P_2, 0), (P_3, \Delta))$ and computes the buffers for each party, where buffer_3 does not include tx^*. Since the simulator only knows the buffers of each party and that one of the three parties has won the selection, but not which one. To create $\mathcal{C}^*$ the simulator will be required to pick one of these buffers. Note that $\mathcal{S}$ *does not know the distribution of the stake*, so it needs to make a decision according to its own strategy. However, fix any strategy for choosing the block proposer, one can show an environment that chooses different stake distributions, that would nullify that strategy.

Hence, the probability of simulating this attack corresponds to the probability p of guessing the party that is eligible to extend the chain. Hence, $Pr[\mathcal{Z} \to 1 | \mathsf{IDEAL}_{\mathcal{S},\mathcal{G}_{\mathsf{PL}}^*}] = p$

Thus the advantage of the environment $\mathcal{Z}$ in distinguishing between the two worlds is $(1 - p) \neq \mathsf{negl}(n)$ implying $\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*} \not\equiv \{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$. Since UC security demands that there should be a single simulator that works for any environment and input distribution, the above protocol is insecure.

4 A Lower Bound on the Anonymity of State Machine Replication

In this section, we describe a lower bound on the anonymity of state machine replication protocols tolerating a f fraction of Byzantine parties. In particular, we show that there exist executions of deterministic state machine replication protocols in a permissioned setting where an honest party does not obtain better than $(1-2f)$-anonymity. In the following section we then show how such a lower bound can be used by an adversary to learn whether an honest party (or a set of honest parties) hold a stake of $(1-2f)$ fraction or higher.

4.1 Definitions

We start with defining terminology required for our lower bound. A state-machine replication protocol provides clients with the same interface as that of a single non-faulty server storing a sequence of values. Typically, these values are stored as blocks, where each block contains a set of transactions, each signed by the clients. We model the sequence of blocks output by a party P_i via a write-once array $[B_1^i, B_2^i, \dots]$ (initialized to $\perp$ values), where B_j^i indicates the j-th block output by P_i.

Definition 1 (State Machine Replication)**.** *Let Π^f be a protocol executed by parties $P_1, \dots, P_n$ who receive transactions as input and maintain a local array B as described above. We say that Π^f is an f-secure z-live state machine replication protocol if the following properties hold when up to an f fraction of parties is Byzantine:*

- *Consistency: If two honest parties P_i and P_j output blocks B_k^i and B_k^j in slot k, then $B_k^i = B_k^j$.*
- *External validity: The output block is either $\perp$ or each transaction included in the output block is signed by a client.*
- *Liveness: Any transaction that is input to an honest z fraction of the n parties is eventually output by every honest party in some block.*

We want to capture the notion of anonymity with state machine replication protocols. The notion of anonymity allows a party sending a message in the protocol execution to remain indistinguishable from a group of other parties. In particular, we define a protocol message as a-anonymous if a party sending a message can be anonymous among an a fraction of all parties. We will now present a definition for a single-shot anonymous state machine replication protocol:

Definition 2 (Single-shot anonymous state machine replication)**.** *Let $\Pi^{f,z,t,a}$ be a protocol executed by parties $P_1, \dots, P_n$ who receive transactions as input and output a block B. We say that $\Pi^{f,z,t,a}$ is an f-secure (z,t)-live a-anonymous single-shot state machine replication protocol if the following properties hold when up to an f fraction of parties is Byzantine:*

- *Consistency: If two honest parties P_i and P_j output blocks B^i and B^j, then $B^i = B^j$.*
- *External validity: The output block is either $\perp$ or each transaction included in the output block is signed by a client.*
- *Liveness: Any transaction that is input to at least an honest z fraction of the n parties more than t time units before the start of the protocol is output by every honest party.*
- *Anonymity: The protocol obtains a-anonymity, i.e., every message sent by an honest party in the protocol is anonymous among an a fraction of parties participating in the protocol.*

Each time unit described in this section equals Δ time. For simplicity, in this section, we always refer to integral time units. Note that the above definition makes the liveness constraint concrete: it specifies a time delay before the start of the execution of the single-shot SMR protocol such that if a z fraction of honest parties receive an input, the protocol should output it. If the value of t is large, the liveness constraint provided by the protocol is weak — only old enough transactions are required to be output by the parties.[2] Observe that $z \leq 1 - f$ is a trivial bound on z since a protocol needs to be live when all honest parties receive a transaction.

4.2 A $(1-2f)$-anonymity Lower Bound

An adversary can control an f fraction of Byzantine parties and can know the state of each of these parties. Hence, $(1-f)$-anonymity is a trivial lower bound on the anonymity that can be obtained for any protocol. In the following, we present a tighter lower bound of $a = (1-2f)$-anonymity for an f-secure (z,t)-live a-anonymous single-shot state machine replication protocol $\Pi^{f,z,t,a}$ when $f < 1/3$.

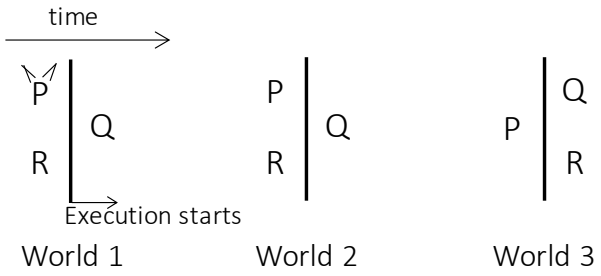

Figure 6: **Intuition for the lower bound.** The figure shows three different execution worlds. In each world, the x-axis represents time and the vertical line represents the start of execution. Parties are split into three groups P, Q, and R of sizes f, f, and $1-2f$ respectively. If a group of parties is placed before the vertical line, then they have received an input v before the start of execution.

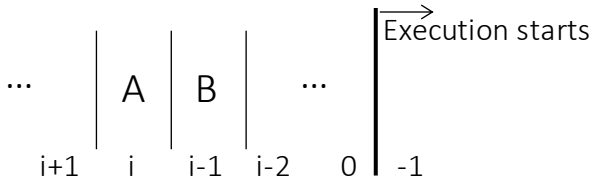

Figure 7: **The notation World $W^{i,i-1}_{A|B}$.** A world representing the times when parties receive inputs. The x-axis represents time units before the start of execution. While times are represented as integers, they should be thought of as integral multiples of Δ. In this world, parties in set A and B receive inputs i and $i-1$ time units before the start of execution.

Intuition. Let us first reason why this holds for $t = 0$, i.e., for a protocol that guarantees an output of v when $(1-f)$ fraction of honest parties have received the same input v before the start of the protocol execution. The value of $a = 1-2f$. Suppose the parties are split into three sets P, Q, and R of sizes f, f, and $(1-2f)$ respectively. If the protocol starts in a world (World 2 in Figure 6) with no faulty parties and with an input v that is sent to parties in P and R, then for the liveness property

[2] We assume that parties in the protocol execution have large enough bandwidth available to them and are as such not constrained by the available bandwidth.

to hold, all honest parties output v. Hence, if the protocol required parties in R (and also P) to send a message that resulted in the output to be v, then the protocol obtains anonymity no better than $1-2f$. This is because in a different world (World 1 in Figure 6), parties in P could be corrupt and honestly following the protocol specification, and hence R does not obtain more anonymity than the size of its set (recall that parties in Q did not obtain the input and R cannot detect that P is corrupt if it follows the protocol specification). Thus, it remains to prove that corrupt P's messages alone does not result in all honest parties to output v in World 2. Observe that if this were the case, then in a different world (World 3 in Figure 6) where only honest P obtains the input, the output will still have been v (since R's contribution did not matter and the protocol is deterministic). However, in this world, P obtains $f < 1-2f$-anonymity for $f < 1/3$.

Formal lower bound. We now present our lower bound for a general t, z, and $f < 1/3$. Throughout, we assume that the parties are split in three groups P, Q, and R that respectively contain f, f, and $1-2f$ fractions of all parties. We use the notation $W^{i,i-1}_{A|B}$ to denote the order in which an input v arrives at sets of parties A and B in a given world (see Figure 7). Specifically, the notation $W^{i,i-1}_{A|B}$ means that in this world, the input v is received by parties in set A and set B at i and $i-1$ units of time respectively, *before* the start of the protocol. To clarify, if a party receives an input $i = 0$ time units before the start of the protocol, the party has received the input when the protocol starts. If $i < 0$, then the party does not have access to the input at the start of the protocol. Also, observe that due to the Δ-synchrony constraint, a group of honest parties can only receive an input at most one time unit (Δ time) after the first honest party receives this input. This explains why the time units i and $i-1$ are always consecutive in any world.

Our lower bound results will require sending only one input to different parties at different times, and hence, the notation $W^{i,i-1}_{A|B}$ suffices. Moreover, since only one input v is sent, the output is either v or a $\perp$. We now prove a sequence of lemmas that help us eventually prove the desired lower bound on anonymity. Please refer to Figure 8 to aid the understanding of Lemmas 1 and 2.

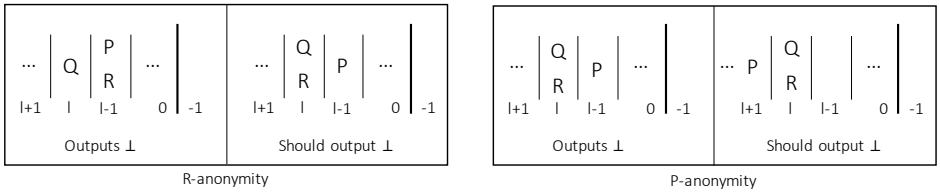

Figure 8: Figure representing the worlds used in Lemmas 1 and 2.

Lemma 1 (R-anonymity)**.** *Assume $\Pi^{f,z,t,a}$ provides better than $(1-2f)$-anonymity and $f < 1/3$. Let $0 \le \ell \le t$. If the output of $\Pi^{f,z,t,a}$ in World $W^{\ell,\ell-1}_{Q|P,R}$ is $\perp$ then the output of Π in World $W^{\ell,\ell-1}_{Q,R|P}$ is also $\perp$.*

Proof. By assumption, the output of $W^{\ell,\ell-1}_{Q|P,R}$ is $\perp$. In World $W^{\ell,\ell-1}_{Q,R|P}$ parties in Q and R receive input v, ℓ time units before the start of the protocol. If the output of $\Pi^{f,z,t,a}$ in $W^{\ell,\ell-1}_{Q,R|P}$ were v then parties in R do not obtain anonymity better than $1-2f$. This is because the protocol is deterministic, and the only aspect that changed is the fact that parties in R received the input earlier. This must be a result of at least one party in R sending a message based on the input v.

This contradicts the claim that $\Pi^{f,z,t,a}$ provides better that $(1-2f)$-anonymity. Therefore, the output of $\Pi^{f,z,t,a}$ in worlds $W^{\ell,\ell-1}_{Q|P,R}$ and $W^{\ell,\ell-1}_{Q,R|P}$ must be $\perp$. □

Lemma 2 (P-anonymity)**.** *Assume $\Pi^{f,z,t,a}$ provides better than $(1-2f)$-anonymity and $f < 1/3$. Let $0 \le \ell \le t$. If the output of $\Pi^{f,z,t,a}$ in World $W^{\ell,\ell-1}_{Q,R|P}$ is $\perp$ then the output of $\Pi^{f,z,t,a}$ in World $W^{\ell+1,\ell}_{P|Q,R}$ is also $\perp$.*

Proof. By assumption, the output of $W^{\ell,\ell-1}_{Q,R|P}$ is $\perp$. The only difference between World $W^{\ell,\ell-1}_{Q,R|P}$ and World $W^{\ell+1,\ell}_{P|Q,R}$ is in P receiving the input $\ell+1$ time units earlier instead of $\ell-1$ time units. If the output of $\Pi^{f,z,t,a}$ in world $W^{\ell+1,\ell}_{P|Q,R}$ is v, then parties in P do not obtain better than $f < 1-2f$ anonymity. This is because the protocol is deterministic, and the only aspect that changed is the fact that parties in P received the input earlier. This must be a result of at least one party in P sending a message based on the input v. This contradicts the claim that $\Pi^{f,z,t,a}$ provides better that $(1-2f)$-anonymity. Hence, the output of $\Pi^{f,z,t,a}$ in World $W^{\ell+1,\ell}_{P|Q,R}$ must be $\perp$. □

Lemma 3 (Q-anonymity)**.** *Assume $\Pi^{f,z,t,a}$ provides better than $(1-2f)$-anonymity and $f < 1/3$. Let $0 \leq \ell \leq t$. If the output of $\Pi^{f,z,t,a}$ in World $W^{\ell,\ell-1}_{P,R|Q}$ is $\perp$ then the output of $\Pi^{f,z,t,a}$ in World $W^{\ell+1,\ell}_{Q|P,R}$ is also $\perp$.*

Proof. Follows similarly to the proof of Lemma 2. □

Lemma 4 (Invariance of Anonymous Output)**.** *Assume $\Pi^{f,z,t,a}$ provides better than $(1-2f)$-anonymity and $f < 1/3$. For any $0 \leq \ell \leq t$, the output of $\Pi^{f,z,t,a}$ in (i) World $W^{\ell,\ell-1}_{Q,R|P}$ is $\perp$, and (ii) in World $W^{\ell,\ell-1}_{P,R|Q}$ is $\perp$.*

Proof. We prove the statement by induction.

Base case: $\ell = 0$**.** We first note that when $\ell = 0$, the output of $\Pi^{f,z,t,a}$ in World $W^{0,-1}_{Q|P,R}$ must be $\perp$. If the output was v, $\Pi^{f,z,t,a}$ would not provide better than $f < (1-2f)$-anonymity to parties in Q. This is because in this world, Q is the only set of parties who have received v before the start of the execution and who could have participated with input v. Hence, the output of $\Pi^{f,z,t,a}$ in $W^{0,-1}_{Q|P,R}$ is $\perp$. Next, it follows from Lemma 1 that the output of $\Pi^{f,z,t,a}$ in World $W^{0,-1}_{Q,R|P}$ is also $\perp$.

Starting with World $W^{0,-1}_{P|Q,R}$ and using a similar argument, we can show that the output in World $W^{\ell,\ell-1}_{P,R|Q}$ is $\perp$.

Inductive step: $\ell = k$**.**

Proposition 1. *Assume $\Pi^{f,z,t,a}$ provides better than $(1-2f)$-anonymity. If the output of $\Pi^{f,z,t,a}$ in worlds $W^{k,k-1}_{Q,R|P}$ and $W^{k,k-1}_{P,R|Q}$ is $\perp$, then the output of $\Pi^{f,z,t,a}$ in worlds $W^{k+1,k}_{P,R|Q}$ and $W^{k+1,k}_{Q,R|P}$ is also $\perp$.*

By inductive hypothesis we know that output of $\Pi^{f,z,t,a}$ in worlds $W^{k,k-1}_{Q,R|P}$ and $W^{k,k-1}_{P,R|Q}$ is $\perp$. Using this hypothesis and respectively applying Lemmas 2 and 3, we can state that the output of $\Pi^{f,z,t,a}$ in worlds $W^{k+1,k}_{P|Q,R}$ and $W^{k+1,k}_{Q|P,R}$ is also $\perp$. Next, we can apply Lemma 1 to state that the output of $\Pi^{f,z,t,a}$ in worlds $W^{k+1,k}_{P,R|Q}$ and $W^{k+1,k}_{Q,R|P}$ is also $\perp$. This proves the proposition. □

Theorem 1. *For any deterministic single-shot anonymous state machine replication protocol $\Pi^{f,z,t,a}$ satisfying Definition 2, there exists an execution that cannot provide better than $a = (1-2f)$-anonymity when $z = 1-f$ and $f \leq 1/3$.*

Proof. We prove by contradiction. Let $\Pi^{f,z,t,a}$ be a protocol that achieves better than $(1-2f)$-anonymity, and (z,t)-liveness.

Consider a World $W^{t,t-1}_{Q,R|P}$ where all parties are honest, and Q and R receive input v, t time units before the start of the protocol, while parties in P receive it $t-1$ time units earlier. Due to the liveness property, the output of the honest parties in $\Pi^{f,z,t,a}$ in World $W^{t,t-1}_{Q,R|P}$ must be v.

Now consider a different World $\hat{W}^{t,t-1}_{Q,R|P}$ that is exactly the same as World $W^{t,t-1}_{Q,R|P}$ except that parties in Q are corrupt. However, the parties in Q follow the protocol specification. Hence, for parties in R (and P), this world is indistinguishable from World $W^{t,t-1}_{Q,R|P}$. Hence, the honest parties in P and R will still output v.

However, this contradicts Lemma 4 which states that, for a $\Pi^{f,z,t,a}$ that provides better than $(1-2f)$-anonymity and $f < 1/3$, for any $0 \leq \ell \leq t$, the output of $\Pi^{f,t,a}$ in World $W^{\ell,\ell-1}_{Q,R|P}$ must be $\perp$. □

Interpreting the lower bound constraints. We now make a few remarks about this lower bound. First, our lower bound statement fixes a liveness constraint, and then argues that a protocol satisfying this constraint cannot obtain anonymity for some execution. While we do not prove the converse, it should hold true too: if we fix an anonymity constraint (for instance, to be $(1-f)$), it should be the case that the protocol will not be live. Second, the arguments in the lower bound assume that parties use an input during the execution only if it was available to them at the start. However, there may exist protocols that use inputs received after the start of the execution. While our lower bound does not directly capture such protocols, the ideas used in the lower bound should be applicable to these protocols too. In particular, there should always exist a point in time during the execution after which a party cannot change its input; if the adversary sends an input to some parties before this time, and to some parties after, then similar arguments as in the lower bound should hold true. Third, the lower bound uses integral units of time for input delivery where each unit of time can be of Δ duration. This is achievable since the adversary is responsible for delivering messages in the protocol (under the Δ-delay constraint), and it can always ensure that the message is delivered at these integral boundaries, e.g., receiving input at $l = 0$ implies that the input is received at exactly the start of protocol execution. Fourth, our lower bound holds for $f \leq 1/3$. For a larger f, the lower bound can be trivially extended to obtain a bound $1 - 2(1/3) = 0.34$-anonymity. Improving the lower bound at a higher f is still an open question. Fifth, while the liveness bound is parameterized with z, the anonymity bound assumes the maximum z value of $1 - f$, i.e., a protocol provides liveness only when all honest parties receive the transaction. The bound is trivially applicable even when the protocol provides stronger liveness guarantees, i.e., when $z < 1 - f$. Moreover, the lower bound can potentially be extended to obtain an anonymity bound of $\min(1 - 2f, \max(f, z - f))$ for a generic z. Finally, the lower bound setup assumes that the start of protocol execution is synchronized among parties. It is unlikely that a protocol that works in a generic unsynchronized setting will fail to work only when the execution starts are perfectly synchronized.

5 Attacks on Anonymity in the PoS setting

Our lower bound in Section 4 shows that the liveness requirement can be leveraged by an adversary to determine the participation of the parties in the protocol. In a PoS setting, the participation of a party in the blockchain protocol is directly related to the stake it owns. In this section, we connect the strategy used to prove the lower bound for state-machine replication in the permissioned setting to a strategy to learn the stake distribution in any PoS blockchain protocol that achieves (z, t)-liveness, where $z = 1 - f$. Here, f is the fraction of the total stake owned by the malicious parties.

We assume the adversary starts with no information about the stake distribution. The adversary aims to establish a configuration as in World $\hat{W}^{t,t-1}_{Q,R|P}$ to test that parties in a set R participated, and hence infer information about their stake. The crux of the attack is to use the (z, t)-liveness condition to identify R and P. In the PoS setting below, we set $t = 0$. This is equivalent to the case of parties receiving the input v before the start of execution of the protocol or after the execution of the protocol.

In Section 5.1 we show how an adversary over a few executions of the protocol can learn exactly a set of parties P and R that hold f and $1 - 2f$ fraction of the total stake respectively. In Section 5.2 we then show how an adversary can target a specific party and estimate the stake owned by the party. This attack can be extended over multiple executions to other parties to learn the entire stake distribution.

Setting input configuration. Consider the case when the protocol is executed at time r. To

emulate the input configuration of $\hat{W}^{t,t-1}_{Q,R|P}$ in our attacks, the adversary needs to make sure that every transaction quickly reaches all parties, except for a specific transaction "v", which is delivered to the parties in P after a delay. To capture this network capability of the adversary we define a function $\mathsf{Delay}(v, S)$ that is available to the adversary. Delay takes as input a value v, and a set of parties S. This function adds a message v to the buffer of all parties in the set $\mathbb{P} \setminus S$, where $\mathbb{P}$ is the set of all parties in the network. We assume that if an honest party has v in its buffer, then the party plays with v in the protocol. Section 6 shows how this function can be implemented in practice.
Setting corrupted Q. For this set, the adversary will corrupt stakeholders until it gains an f fraction of the total stake.
Assumption on stake In our attacks we require that the stake of the honest parties do not change drastically in each execution of the protocol. For simplicity of presentation we assume that the stake of the participants remains constant.

5.1 The Tagging Attack

Let $\mathbb{P}$ be the set of all parties. In this attack, the goal of the adversary is to divide the set $\mathbb{P}$ into sets P, Q and R such that they have f, f and $1-2f$ fractions of the total stake. As noted earlier, the set Q is corrupt, therefore the adversary already knows a set of parties that together make up f fraction of the total stake.

As a warm-up, let us first consider a world, where the adversary does not corrupt anyone. The goal of the adversary here is to divide the set of parties $\mathbb{P}$ into two sets of P and R such that they contain f and $1-f$ fraction of the total stake. Initialize R as $\mathbb{P}$ and P as $\emptyset$. As noted earlier using the Delay function, an adversary is able to control the buffers of parties. More specifically, the adversary is able to ensure that some parties receive a value v after a certain delay. The strategy of the adversary to populate P and R is as follows: Pick a random $P_i \in \mathbb{P}$ and add it to the set P and delete it from the set R. Create a valid input v and call the function $\mathsf{Delay}(v, P)$. Let the current execution of the protocol be for a round r. If the output includes v, the adversary knows that P_i has stake less than f. Now the adversary picks another P_i adds it to R and repeats the above attack with a different v. The adversary repeats this process until some sets P and R results in an output without v. The adversary then outputs P as a set that has at least an f fraction of the total stake and R as a set that has at most an $1-f$ fraction of the total stake. Why? By the definition of liveness, if a value v is not in the output of a protocol, it must mean that v was in buffers of parties with $< 1-f$ fraction of the total stake.

We call this *the tagging attack*, since an adversary in some sense tags the inputs of parties and observes the output to see if the tagged inputs appear or not.

Tagging Attack

We describe the attack from the point of view of the adversary. The adversary knows the set of parties $\mathbb{P}$ and initializes the set $R = \mathbb{P}$ and $P = \emptyset$. The goal of the adversary is to populate P and R, such that they have at least f and $1-f$ fraction of the total stake. We denote an adversarially created value with the variable v_r where r is the current round. While the output of execution at round r includes v_r

1. Pick an arbitrary party P_i and do $P = P \cup \{P_i\}$ and $R = R \setminus \{P_i\}$
2. Create a valid input value v_r and run $\mathsf{Delay}(v_r, P)$ and update $r = r + 1$

Output R and P

Figure 10: Tagging Attack with no corruption

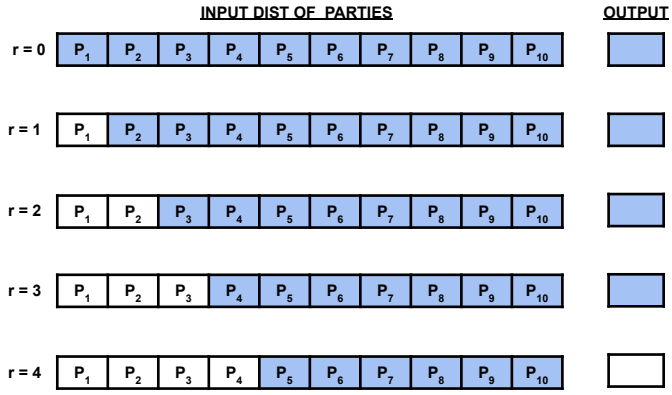

Figure 9: The tagging attack. Let $\mathbb{P} = \{P_1 \dots P_{10}\}$ be the set of parties and each row represent the input distribution and the output with this distribution. If an adversarial value is in the buffer of a party we color the box blue. One by one the adversary adds a party P_i to a set P and observes output. Finally when output is white, the adversary knows that the set P in round $r = 4$ has at least f fraction of the total stake and the rest of the parties that constitute R has less than $1 - f$ fraction of the total stake

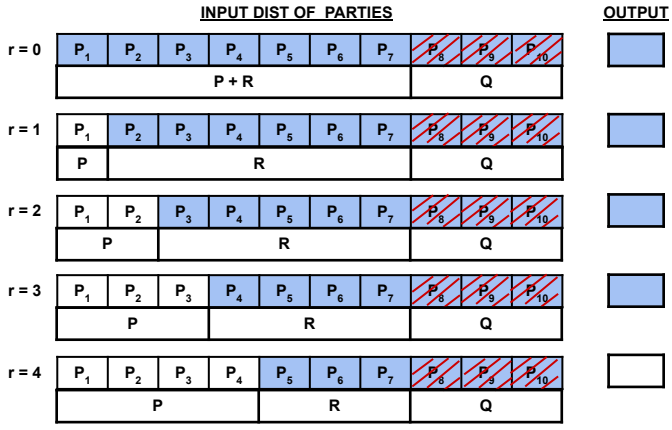

Figure 11: The tagging attack with corrupted parties. Let $\mathbb{P} = \{P_1 \dots P_{10}\}$ be the set of parties and each row represent the input distribution and the output with this distribution. Let the adversary control $Q = \{P_8, P_9, P_{10}\}$ (boxes with the red stripes). If a view is set by the adversary we color the box blue. In every execution of the protocol the adversary plays with the blue view. Initially all parties start having blue, then the output is blue. The adversary one by one adds parties to the set P, sets the input of P as white and observes the output. If the output is white, the adversary knows that the set R has at most $1 - 2f$ fraction of the total stake and set P has atleast f fraction of the total stake.

Now let us consider the case where the adversary manipulates the way the corrupted set Q participates. The adversary mounts exactly the same attack as above. But in every execution it instructs the parties in Q to play with v. Now when the output does not include v, the adversary learns that the set R will consist of parties that have at most $1-2f$ fraction of the total stake. And the remaining parties, i.e. $\mathbb{P} \setminus \{Q \cup R\}$ make up P. Why? By the definition of liveness if the output does not include v, it implies that the set $Q \cup R$ does not make up $1-f$ of the total stake. Since the adversary also knows the set Q that makes up f fraction of the total stake, the adversary concludes that R is a set with at most $1-f$ total stake and therefore $P = \mathbb{P} \setminus \{Q \cup R\}$ makes up at least f fraction of the total stake. We present a visual example of the attack in Fig 11.

5.2 The Reverse Tagging Attack

In this section we present a targeted attack where an adversary is able to precisely estimate the stake owned by a certain party, Alice. We denote the stake owned by Alice as $\mathsf{stake}_{\mathsf{Alice}}$.

In this attack, the adversary's strategy is to deliver the inputs as follows. It delivers an input v to every party, except Alice. The adversary now observes the execution of the protocol where Q plays with v as well. If the output is not v, the adversary can conclude that $\mathsf{stake}_{\mathsf{Alice}} > f$. If the output is v the adversary learns that the $\mathsf{stake}_{\mathsf{Alice}} \leq f$. The adversary now repeats the process above except that only $f-d$ fraction of Q play with the value v. If the output is not v, the adversary learns that $\mathsf{stake}_{\mathsf{Alice}} > f-d$ fraction of the total stake. Why? If the output is not v by the definition of liveness it is implied that $< 1-f$ fraction of the total stake played with input v. Now this implies, $((1-f) - \mathsf{stake}_{\mathsf{Alice}}) + (f-d) < 1-f$, which implies $\mathsf{stake}_{\mathsf{Alice}} > f-d$. On the other hand, if the output is v then the adversary runs the process again in the next round with a larger d. We note than an exponential search gives the best results in estimating the stake of Alice. We present a visual example of the attack in Fig 12.

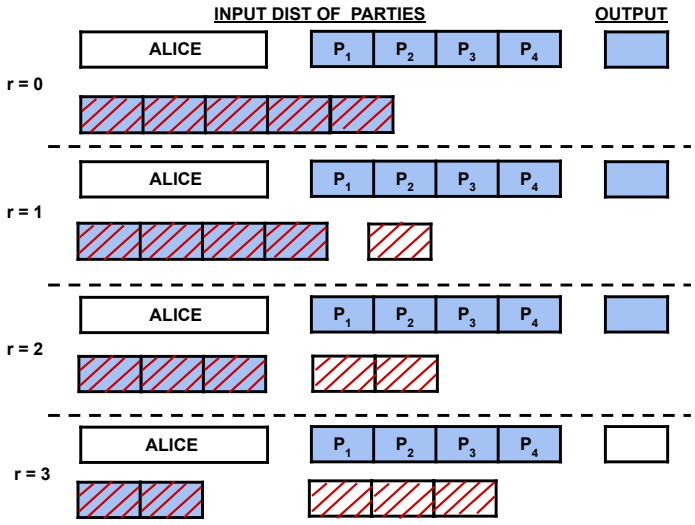

Figure 12: Example of reverse tagging attack. Each r includes the input distribution of the honest parties (first row: Alice and $P_1 \dots P_4$), the malicious parties (second row: boxes with the red stripes) and an output. If a view is set by the adversary we color the box blue. The adversary wants to find out Alice's stake. In round $r = 1$, the adversary gives $P_1 \dots P_4$ the blue view. The adversary plays in the protocol with a blue view with a smaller fraction than f. It observes the output and sees it is blue. In this case the adversary learns nothing. The adversary repeats this until $r = 3$ when it observes the output is white. Now the adversary can conclude that Alice has more stake than the fraction of adversarial stake that played with blue.

We note that the adversary could do this with every honest party until it learns the exact stake distribution of all parties.

Reverse Tagging Attack

Let the current round be r. We denote the adversarially created value with a variable v_r.
The adversary wants to determine $\mathsf{stake}_{\mathsf{Alice}}$. The adversary initializes a fraction d to 0, where $f - d$ is adversary's guess of $\mathsf{stake}_{\mathsf{Alice}}$.
While the output of execution at round r includes v_r:

1. Update $d = d + \frac{1}{\mathsf{totalStake}}$ and create a valid input v_r and run $\mathsf{Delay}(v_r, \mathsf{Alice})$.
2. Instruct parties in Q that make $(f - d)$ fraction of total stake to play with v_r. Update $r = r + 1$

Output $(f - d)$ as the estimate of $\mathsf{stake}_{\mathsf{Alice}}$.

Figure 13: The Reverse Tagging attack

Remark (On deterministic protocols)**.** *Our lower bound in the previous section is only for* deterministic *protocols that guarantee (z, t)-liveness. However, protocols may use randomness for various reasons such as cryptographic operations, efficiency, and anonymity. Since protocols such as Algorand [4] and Ouroboros [3, 8] use randomness for efficiency purposes (e.g., deciding the next committee members), the above described attacks work so far as the protocols guarantee liveness. In general, while our lower bound is constrained to deterministic protocols, the ideas described may extend to randomized protocols too unless the protocols use randomness specifically to obtain anonymity.*

6 Practicality of Our attacks

In this section we describe how our attacks can be carried out on a real-world blockchain network. There exists no implementation of a privacy-preserving PoS blockchain, therefore we describe our attacks for a privacy-preserving proof-of-work blockchain, namely the Zcash blockchain [12].

In Section 6.1 we present an overview of how the Zcash p2p network works. We then describe (in Section 6.2) how to implement the Delay function defined in Section 5. We note that it suffices to show an implementation of this function to mount the tagging (Section 5.1) and reverse tagging attacks (Section 5.2).

6.1 The Zcash peer-to-peer network

Establishing connections between nodes. When a Node joins the Zcash network it needs to connect to existing nodes on the network. To establish these connections the Node initiates a TCP handshake with these nodes. To achieve this, the only information the Node needs is the IP address of the network nodes. At the end of this process the Node will have established incoming and outgoing connections to $\mathsf{Peer}_1 \ldots \mathsf{Peer}_k$.

Receiving and sending transactions. Nodes on the Zcash p2p network follow a three-step protocol to propagate transactions. To send a transaction to a peer, the Node first sends just the transaction hash to $\mathsf{Peer}_1, \ldots, \mathsf{Peer}_k$ and will follow up with the entire transaction only if it is requested. In more detail propagating a transaction across the Zcash p2p network involves the following steps:

- **Inventory step:** In this step the Node announces the knowledge of a tx to its peers Peer_i. The

Node sends an INV message which contains H_{tx} to its peers. If this hash was observed before, Peer_i simply ignores the INV message, else it proceeds to the *get data step*.

- **Get data step :** Peer_i sends a command GETDATA, H_{tx} to Node to request the transaction tx. Note that if a Peer_i has requested GETDATA for a particular INV message from Node *then it will ignore* INV *messages for the same transaction from other peers for a specific amount of time (2 min in Zcash*[3]*) and simply add those* INV *messages to a queue* This time-out serves as a window for the Node to respond to the GETDATA message, and will be relevant to our attack.

- **Send** tx **:** In this step Node responds with the tx for the corresponding INV it sent in the first step. The Node then adds this transaction to its buffer.

6.2 Implementing Delay

We assume that the adversary already knows the IP address of a victim, let us denote such a node by victimNode. We assume that the victimNode allows incoming connections and the adversary connects to the victimNode using the process described above.

Recall that in the Delay function the goal of the adversary is to deliver a message to all nodes except the victimNode. Delay can be implemented using a procedure called *Invblocking*, first described in [13] and also used in [30]. For completeness we present the strategy here: 1) Create a transaction tx and compute $(\texttt{INV}, H_{\mathsf{tx}})$. 2) Send $(\texttt{INV}, H_{\mathsf{tx}})$ to the victimNode and all the adversary's peers 3) Upon receiving GETDATA from all peers, respond to all peers with tx immediately except for the victimNode

The victimNode now waits for a time-out until it requests a GETDATA from another peer who also sent an INV with the same H_{tx}. Thus the adversary succeeds in delaying the receipt of tx to the victimNode. Note that if an adversary has multiple connections to the victimNode it could delay the transaction until the victimNode sends GETDATA to an honest peer. In Zcash, a block is mined approximately every 75 seconds. Since the time-out is 2 minutes in Zcash, the *Invblocking* attack allows the adversary to ensure a transaction is not in the buffer of a victimNode for an entire round, which is what we need to implement Delay.

Anonymous communication. The most popular anonymous communication protocols to improve privacy of nodes in cryptocurrencies like Bitcoin, Zcash and Monero are either Tor [14] or the recently proposed protocol Dandelion [15]. We note that as long as an adversary can connect to a victimNode either through Tor relays or a direct connection in the case of Dandelion, the adversary can mount the *Invblocking* attack described in the previous section. This is so because the victimNode running the Zcash daemon will always wait for a time-out until it requests a GETDATA from a different peer. We note in the case that the node is set up as Tor hidden service the attack only results in the adversary learning the OnionCat address that is used to represent the Tor hidden service. Moreover, according to [16], the situation might actually be worse. In their work, they present techniques to fully control the view of a victimNode by leveraging DoS prevention mechanisms of the Bitcoin software. The same DoS prevention mechanisms exist in Zcash as well, and could be used by an adversary to control the victimNode's view.

7 Suggested Mitigation

In this section we discuss strategies that could mitigate or prevent the de-anonymization attacks described above. Recall that our attacks are based on the fact that a network adversary can provide different views to different parties and then, based on the protocol output, determine which parties participated in the protocol. To defend against this attack, an effective countermeasure would be to ensure that all parties get the same view. This can be done in two ways: 1) At the application

[3] See Line 2171 of https://github.com/zcash/zcash/blob/master/src/net.cpp

level, by combining views received by honest parties across the network. 2) At the network level, by leveraging reliable anonymous broadcast.

7.1 Sanitization on the application layer

A potential idea is to introduce an external protocol, that we call *sanitization* protocol, that parties can use to get a view of the network that is not poisoned by targeted delay attacks. The sanitization protocol, would be executed by a set of parties, the *sanitizers*, whose role is to collect transactions from the network and jointly compute the sanitized view. When a stakeholder wants to participate in a PoS protocol, instead of using its local view of the network, it will use the sanitized view. This idea has been proposed in the past. Apostolaki et al. introduced SABRE [31], a Bitcoin relay network which relays blocks through a set of dedicated nodes, that are resilient to routing attacks. Their goal was to prevent routing attacks on Bitcoin that allows AS-level attackers to partition the Bitcoin network. While their work was not intended to protect the anonymity of Bitcoin players, we observe that if the SABRE infrastructure existed, it could serve to run the sanitization protocol. In SABRE [31], the relay nodes are trusted and fixed. To remove this trust assumption an idea would be to rely on the honest majority of stake in a PoS blockchain and select nodes from the network. This gives us a guarantee that a majority of the selected nodes will be honest and we will not need to make any new trust assumptions. Unfortunately, this approach does not work. Why? Selecting sanitizers based on stake jeopardizes the stake privacy of the sanitizers, since by participating in the "sanitization process" they can be targets of the very same attacks we have shown in the previous sections. Moreover, if the sanitizers are not fixed and new sanitizers are picked in every round, the adversary might even learn information on the stake distribution of the network.

Thus, if we trust the sanitizer nodes we can circumvent the lower bound in Section 4 since the information is always sourced from a trusted set of ndoes and an adversary is no longer able to set inputs for specific parties. On the other hand, if we assume that the sanitizers are selected based on stake, our attacks will still hold. We leave it to future work to investigate an implementation of sanitization with other assumptions that can circumvent the lower bound. To summarize, our proposed approach based on sanitization has the following properties: **Pros.** It protects the privacy of the participating parties in the chain extension protocol (i.e., the block proposers). **Cons.** Sanitizers lose their privacy and need to be trusted.

7.2 Reliable Broadcast Mechanisms

One way to ensure all parties have the same view is to somehow enforce that a message sent over the network is "received" by all parties at the same time. As outlined in the introduction, such guarantee cannot be promised by gossip protocols since an adversary can delay messages arbitrarily or block certain messages from reaching a target party (or set of parties). Reliable broadcast [32] [33] is a protocol that has an agreement guarantee which says that if any correct node delivers m, then every correct node delivers m. That is, even if a malicious party "sends" a message to a party it wont be considered "delivered" until all honest parties have received it, thereby *virtually* setting $\Delta = 0$. If realized this functionality would circumvent our lower-bound since, with no delay, the adversary cannot split parties into subsets with differing views.

Unfortunately, existing candidates for reliable broadcast protocols require *direct* connections between all parties in the protocol and hence is not scalable. Recent works, such as Blinder MPC [34] and PowerMix [35] propose a server-client model where the direct connection is not required among all parties; rather a client need only to be connected to a smaller number of servers.

Unfortunately, these protocols are also not scalable: for a network of N clients and n servers, PowerMix [35])require ($O(\log^2(N))$ server-server rounds of communication, while Blinder MPC [34] requires $O(n \cdot \sqrt{N})$ computation overhead for each client.

Furthermore, in the server-client model, a network adversary might be able to delay the communication between a client and its servers, and hence still mount the attacks described in this paper. **Pros**

If implemented, reliable broadcast channel virtually sets $\Delta = 0$ ruling out our delay-based attacks.
Cons Existing protocols are not scalable and would not be practical in a permissionless blockchain setting.

8 Conclusion

In this work we showed that even if a PoS blockchain protocol is designed to preserve anonymity in the application layer, a network adversary that can control the delivery of messages can de-anonymize parties and even learn the stake distributions of parties in the system. We suggest some countermeasures but each come with their own limitations and require more thought and work. We hope that our findings lead to new discussions on technologies to preserve anonymity of stakeholders against any network adversary.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," no. 2012, p. 28, 2008. [Online]. Available: https://bitcointalk.org/index.php?topic=321228.0

[2] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 913–930.

[3] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[4] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019.

[5] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.

[6] S. Noether, "Ring signature confidential transactions for monero." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1098, 2015.

[7] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Z: Enabling decentralized private computation."

[8] T. Kerber, A. Kiayias, M. Kohlweiss, and V. Zikas, "Ouroboros crypsinous: Privacy-preserving proof-of-stake," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 157–174.

[9] C. Ganesh, C. Orlandi, and D. Tschudi, "Proof-of-stake protocols for privacy-aware blockchains," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 690–719.

[10] F. Baldimtsi, V. Madathil, A. Scafuro, and L. Zhou, "Anonymous lottery in the proof-of-stake setting," in *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*. IEEE, 2020, pp. 318–333. [Online]. Available: https://doi.org/10.1109/CSF49147.2020.00030

[11] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *Annual International Cryptology Conference*. Springer, 2017, pp. 324–356.

[12] E. C. Company, "Zcash release v4.1.1." [Online]. Available: https://github.com/zcash/zcash/releases/tag/v4.1.1

[13] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering bitcoin's public topology and influential nodes."

[14] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[15] G. Fanti, S. B. Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller, and P. Viswanath, "Dandelion++ lightweight cryptocurrency networking with formal anonymity guarantees," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 2, pp. 1–35, 2018.

[16] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 122–134.

[17] 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 *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 127–140.

[18] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and privacy in social networks*. Springer, 2013, pp. 197–223.

[19] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 6–24.

[20] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using p2p network traffic," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 469–485.

[21] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 15–29.

[22] G. Fanti and P. Viswanath, "Deanonymization in the bitcoin p2p network," in *Advances in Neural Information Processing Systems*, 2017, pp. 1364–1373.

[23] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "P2p mixing and unlinkable bitcoin transactions." 2017.

[24] A. Biryukov, D. Feher, and G. Vitto, "Privacy aspects and subliminal channels in zcash," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1813–1830. [Online]. Available: https://doi.org/10.1145/3319535.3345663

[25] A. Biryukov and S. Tikhomirov, "Deanonymization and linkability of cryptocurrency transactions based on network analysis," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 172–184.

[26] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security?" in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 92–102.

[27] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2019, pp. 649–678.

[28] D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 108–126.

[29] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[30] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee, "Txprobe: Discovering bitcoin's network topology using orphan transactions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 550–566.

[31] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "SABRE: protecting bitcoin against routing attacks," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/

[32] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[33] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. IEEE, 2005, pp. 191–201.

[34] I. Abraham, B. Pinkas, and A. Yanai, "Blinder: Mpc based scalable and robust anonymous committed broadcast," Cryptology ePrint Archive, Report 2020/248, 2020, https://eprint.iacr.org/2020/248.

[35] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 887–903.