Public Randomness Extraction with Ephemeral Roles and Worst-Case Corruptions

Jesper Buus Nielsen[1][0000−0002−7074−0683], João Ribeiro[2][0000−0002−9870−0501], and Maciej Obremski[3][0000−0003−4174−0438]⋆

[1] Aarhus University, Aarhus, Denmark
jbn@cs.au.dk
[2] Carnegie Mellon University, Pittsburgh, PA, USA
jlourenc@cs.cmu.edu
[3] National University of Singapore, Singapore, Singapore
obremski.math@gmail.com

Abstract. We distill a simple information-theoretic model for randomness extraction motivated by the task of generating publicly verifiable randomness in blockchain settings and which is closely related to *You-Only-Speak-Once (YOSO)* protocols (CRYPTO 2021). With the goal of avoiding denial-of-service attacks, parties speak only once and in sequence by broadcasting a public value and forwarding secret values to future parties. Additionally, an unbounded adversary can corrupt any chosen subset of at most t parties. In contrast, existing YOSO protocols only handle random corruptions. As a notable example, considering worst-case corruptions allows us to reduce trust in the role assignment mechanism, which is assumed to be perfectly random in YOSO.
We study the maximum corruption threshold t which allows for unconditional randomness extraction in our model:

- With respect to feasibility, we give protocols for t corruptions and $n = 6t + 1$ or $n = 5t$ parties depending on whether the adversary learns secret values forwarded to corrupted parties immediately once they are sent or only once the corrupted party is executed, respectively. Both settings are motivated by practical implementations of secret value forwarding. To design such protocols, we go beyond the committee-based approach that is sufficient for random corruptions in YOSO but turns out to be sub-optimal for chosen corruptions.
- To complement our protocols, we show that low-error randomness extraction is impossible with corruption threshold t and $n \leq 4t$ parties in both settings above. This also provides a separation between chosen and random corruptions, since the latter allows for randomness extraction with close to $n/2$ random corruptions.

1 Introduction

Publicly verifiable randomness is a fundamental resource for many tasks, including contract signing, electronic voting, and anonymous communication and

⋆ The author ordering is randomized. A certificate of the randomization procedure can be found here.

browsing [7,10]. However, such sources of randomness are hard to come by in the wild, and so it is imperative to develop protocols which allow multiple mutually distrusting parties, each with access to their own source of randomness, to agree on a public string of nearly unbiased random bits even under adversarial behavior. We propose and study a simple information theoretic model for extracting randomness which is inspired by the problem of generating publicly verifiable randomness in blockchain settings, sometimes also known as blockchain randomness beacons. Our model is in particular motivated by the notions of *player-replacable* protocols as introduced by Micali [16] and *You-Only-Speak-Once* (YOSO) protocols as introduced by Gentry, Halevi, Krawczyk, Magri, Nielsen, Rabin, and Yakoubov [11]. In these classes of protocols, each party only sends messages once, and therefore need not have a mutable secret state. This gives high resilience to Denial-of-Service (DoS) attacks as often desirable in blockchain settings.

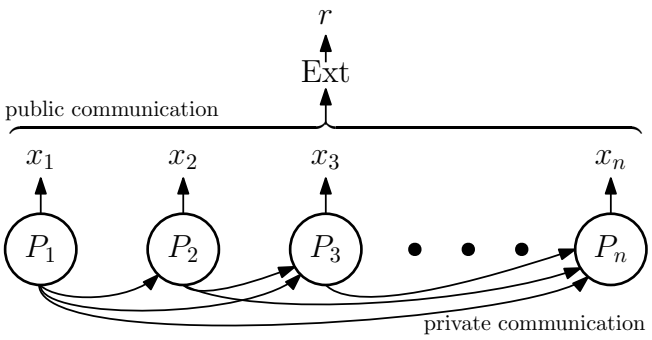

Fig. 1. Illustration of the model. Each party sends a public message and secret messages to future parties. The extracted value r depends deterministically on the public values.

More precisely, we consider a multiparty computation model where n parties $P_1, \ldots, P_n$ are activated sequentially, with each party having access to an internal source of randomness. To begin with, P_1 is executed and it outputs a public value x_1 which we think of as being shown to all parties, including the adversary. Moreover, P_1 also gets to send secret values $s_{1,j}$ to each future party P_j. When a future P_j is executed it receives all previous public values x_i for $i < j$ along with all secret values intended for P_j. We can think of P_i as being described by a distribution D_i defining the conditional probabilities

$$\Pr_{D_i}[x_i, s_{i,i+1}, \ldots, s_{i,n} \,|\, x_1, \ldots x_{i-1}, s_{1,i}, \ldots, s_{i-1,i}] \ .$$

After all parties speak, the goal is to deterministically extract nearly unbiased public randomness from the public values $(x_1, \ldots, x_n)$. The model is illustrated in Fig. 1. We interpret these parties $P_1, \ldots, P_n$ as *ephemeral roles*. In a practical setting there would be a ground set of N parties, usually with $N \gg n$, and

executing each of the n roles entails sampling a party in some manner from the ground set to follow the instructions. This is discussed in more detail in Section 1.1.

We study threshold static worst-case corruptions: The adversary is allowed to corrupt an arbitrary unknown subset of up to t parties before the start of the protocol, where we call t the *corruption threshold*. The adversary sees all public outputs when they are produced. Additionally, it also sees all secret inputs for corrupted parties and can determine their outputs. I.e., when a corrupted party P_i is about to be executed, it is the adversary who receives inputs $(x_1, \dots x_{i-1}, s_{1,i}, \dots, s_{i-1,i})$ and determines the outputs $(x_i, s_{i,i+1}, \dots, s_{i,n})$. Note that the adversary also has access to information it gathered in the past. For example, it also has access to the secret values sent to past corrupted parties $P_{i'}$ with $i' < i$.

Information may be revealed to the adversary in different ways. We consider two scenarios, which we term the *sending-leaks* and *execution-leaks* settings. In the sending-leaks setting, if party P_i is honest and sends the secret value $s_{i,j}$ to a corrupted party P_j during its execution, then the adversary learns $s_{i,j}$ immediately. In contrast, in the execution-leaks setting the secret value $s_{i,j}$ would only be revealed to the adversary later when the corrupted party P_j is executed. The motivation behind these settings is related to how the forwarding of the secret values is implemented in practice. We discuss this in more detail in Section 1.1.

We consider unconditional security only. More precisely, after the protocol ends and some randomness r has been extracted from $(x_1, \dots, x_n)$, we require that r is statistically close to uniform over $\{0, 1\}$ given the view of the adversary. Note that this view includes all secret messages sent or received by corrupted parties. Furthermore, we may generate more random bits by running the protocol several times in parallel. The following general question arises naturally:

> *What is the maximum corruption threshold that allows unconditional randomness extraction in each of the corruption models?*

A Naive First Approach via Standard Multiparty Computation. Our setting is closely related to previous multiparty computation (MPC) models. In particular, it can be seen as a special case of information theoretic MPC of a random string in the models of [8,2]. There is, however, one significant difference: In standard MPC a party can have identity over time, while in our case each party speaks only once. This is also related to the notion of proactive security [13] where refreshment is used to decouple future states of a party from previous states.

Naively, a standard r-round MPC protocol tolerating that t out of m parties can be corrupted immediately yields a protocol in our model where t out of $n = rm$ parties $P_{1,1}, \dots, P_{1,m}, \dots, P_{r,1}, \dots, P_{r,m}$ can be corrupted. This is accomplished by implementing the behavior of the i-th party over the r rounds using r distinct parties $P_{1,m}, \dots, P_{r,m}$ in our model. Then, we use the secret val-

ues to pass the current state of $P_{i,\rho}$ to $P_{i,\rho+1}$, and consider party P_i corrupted if *at least one* of the parties $P_{i,1}, \ldots, P_{i,r}$ is corrupted.

In particular, one can generate unbiased randomness using this approach by having all parties run a verifiable secret sharing protocol of a random value, and then reconstruct all values and XOR them. Using, for instance, the protocol from [9] with 2-round sharing and 1-round reconstruction procedures tolerating $t < m/3$ corruptions, we immediately obtain a randomness extraction protocol in our model tolerating $t < n/9$ corruptions. However, this is not very satisfactory. In fact, this approach implies a reduction by a factor of r in the tolerated corruption threshold. This is inevitable because, in our model, the adversary can concentrate all t corruptions wherever it wishes (and this also applies to proactively secure protocols). Therefore, if the original protocol has some notion of "rounds", then the adversary can concentrate all corruptions in a single round. We find it interesting to understand how the ability to "concentrate" corruptions affects the feasibility of MPC. This is in particular interesting in player-replaceable and YOSO style protocols where there is no identity.

The naive approach above does show that we can withstand *some* constant threshold of corruptions. On the other hand, it is easy to see that we cannot tolerate $t \geq n/2$ corruptions. With this in mind along with the loss incurred by translating a round-based protocol to our setting, we are interested in the following concrete question:

> *Can we improve the lower and upper bounds on the exact maximum corruption threshold that allows for public randomness generation in our model?*

1.1 The Motivation behind our Setting

Our model is very closely related to the *You Only Speak Once (YOSO)* model proposed in [11], which was in turn inspired by the model from [3] and is related to the *fluid MPC* model in [9]. The YOSO model is also related to the notion of a *player-replacable* protocol as introduced by Micali [16]. The main difference between player-replaceable and YOSO protocols is that player-replaceable protocols only allow public messages and YOSO protocols allow secret messages.

The YOSO model considers information theoretic MPC where each party speaks only once and has secure channels to future parties. Therefore, our model is a specialisation of the YOSO model to public randomness extraction. The YOSO model is inspired by the problem of doing MPC in an open blockchain setting. In such a setting, once someone sends a message, their IP address becomes known and they are subject to DoS attacks. To mitigate this issue, blockchain protocols are often designed such that each party speaks only once and does not have to keep state.

For instance, in Bitcoin the next block is produced by a random miner, and so the adversary cannot target a DoS attack on the next party to act. However, there are two additional issues to keep in mind when performing MPC in this model: *First, how does a party send a secret value to an unknown future speaker? Second, how impartial is the selection of speakers?*

The Motivation behind the Sending-Leaks and Execution-Leaks Models. In [3] and [12] so called *role assignment* protocols are proposed to solve the first problem above. In these protocols an ephemeral public key pk for a future role R is made public and at the same time some random party P learns the secret key sk. A role is simply the description of some future part of the protocol, such as "execute the code of party P_{42}". In order to send a secret message s to P, one can then simply broadcast an encryption of s. In the context of a blockchain, the broadcast could for instance be done by posting the ciphertext on the blockchain along with the public value. When it is P's turn to execute the role R in the protocol, it will decrypt the incoming values, compute its outgoing values, and then post them all on the blockchain in one flow. In particular, P speaks only once and does not have to reveal its identity until it sends its values.

A different approach to role assignment is given in [5]. Values to role R are encrypted to identity R in a threshold identity-based encryption scheme, and a rolling committee holds a secret sharing of the master secret key. Once R is about to be executed, the secret key for role R is reconstructed to a random hidden party P which then executes the role.

A role assignment protocol can also be used to implement our model on top of a blockchain to yield public randomness extraction in the YOSO model. Note that the role assignment protocols in [3,12] corresponds roughly to what we call the sending-leaks model. If P is corrupted, then it learns sk once the ephemeral keys pk has been made public. So the adversary learns the secret values forwarded to P as soon as they are sent. The role assignment protocol in [5] corresponds roughly to the execution-leaks model, as P does not learn the identity secret key until execution time. Therefore, both of our models can be motivated by practical settings.

Another important motivation for our model is that it is a clean setting for studying public randomness extraction techniques. This model is simple and detached from its blockchain motivation, and can therefore hopefully draw in researchers from other areas which are not necessarily interested in the nitty-gritty practical details of blockchains. At the same time, it is close enough to its motivating setting that new insights in the model hopefully can lead to better protocols for practical generation of randomness on public blockchains.

Worst-Case vs. Random Corruptions. The YOSO paper [11] studies mainly *random* corruptions (i.e., each party is independently corrupted with some constant probability). The motivation behind this is that, since roles R are assumed to be mapped uniformly to parties P by some perfectly random role assignment mechanism and the adversary in practice can do chosen corruptions of *parties*, then the adversary is *de facto* restricted to random corruptions of roles. As a result, one can morally restrict attention to protocols secure against random corruptions only, and then compile them to practice using role assignment.

However, there are natural motivations for considering worst-case corruptions in this model, as opposed to random corruptions:

1. **Reducing trust in the role assignment mechanism:** The first main reason for considering worst-case corruptions is that the role assignment mechanism might not be perfectly random, in which case the YOSO protocols for random corruptions are no longer secure. Assuming worst-case corruptions in our protocol design allows us to withstand bias on the part of the role assignment mechanism without losing security, and so the required level of trust in this mechanism is greatly reduced. Furthermore, studying chosen corruptions may also inspire more efficient techniques for the intermediate case where role assignment is neither perfect nor extremely biased.
   An example of a role assignment which is highly biased is the *fluid MPC* model in [9]. Here, parties may come and go at will. One motivating setting is parties lending their machines to MPC when they are idle. In this case, honest parties might be seen as registering at random times, but corrupted parties can strategically choose when they join the computation. This is closer to chosen corruptions than random corruptions.
   Another example of imperfect random role assignment is the *A Practically Appealing Weak Batch-RPIR* scheme in [12]. Here, parties are put into small buckets using a known distribution. Then, the next random role is assigned by chosen a random unknown party from the next bucket. This gives a lot of information on which parties could execute which roles, but it is good enough for electing large committees with enough honest parties, which is all that is needed for committee based protocols.
2. **Randomness for small groups:** Another case where it makes sense to consider chosen corruption is when generating randomness for small groups. For instance, this happens naturally in the player simulation technique in secure MPC [14], where a constant sized group carries out a given task. If enough group members are honest, we want the task to be carried out securely. On the other hand, if too many are corrupted we simply count the group as fully corrupted. Overall, security holds if enough groups are honest. In such a setting, random corruptions will have all possible subsets of the group be corrupted with constant probability, and so we might therefore as well study chosen corruptions. We would be interested in under which corruptions the group can generate good joint randomness.
3. **Moving away from established round-based techniques:** Considering only random corruptions as in YOSO [11] ensures that one can stay mostly within established methodologies for round-based MPC, which we have seen do not work well in our setting with worst-case corruptions. Indeed, the following design pattern works for random corruptions: If there is an r-round protocol with committees of size m, then pick m large enough so that if there are less than $(1/2-\delta)m$ corruptions, for a small constant $\delta > 0$, then also among any m parties in a given committee the frequency of corruption will be below half. Following this, design a protocol tolerating less than half corruptions in each committee. If we assign committees to rounds, this leads to a model with rounds where one can assume honest majority during all rounds. This makes it easier to design protocols as one does not have to deal with the *concentration* problem discussed above.

In [11] there is a YOSO protocol for public randomness generation for *random corruptions* with threshold $t = (1/2 - \delta)n$ for any constant $\delta > 0$. This protocol is obtained by first giving an MPC protocol for this setting and then noting that randomness generation is a special case of MPC. The MPC protocol uses a number of committees linear in the security parameter and all committees need to have honest majority. This means that if the protocol is cast in our model with chosen corruptions, then only a vanishing fraction of corruptions can be tolerated since the adversary could concentrate corruptions on a single committee. This gives a worse bound than the previously discussed naive approach via verifiable secret sharing using [9].
With the above in mind, another main motivation for studying chosen corruptions, as opposed to random corruptions, is that it forces us to develop new techniques beyond the committee-based methodology in order to minimize the total number of parties needed for a given task. We hope that studying our proposed model with chosen corruptions can shed light on the following question:

Can we develop YOSO extraction/MPC techniques qualitatively different from the ones developed for the committee-based methodology?

As we shall discuss in more detail in Section 1.4, we also take the committee-based approach as a starting point for our protocols, but then make additional improvements beyond the naive approach by exploiting the structure of the YOSO model.

1.2 Other Related Work

Some papers have also studied the generation of unbiased randomness from a blockchain based on computational assumptions – see, for instance, ALBA-TROSS [7] and the citations therein. This protocol again assumes rounds with at least honest majority in each round, and therefore do not have to deal with the concentration problem.

Other works have considered related models where parties publish public values in sequence but future secret values are not permitted, such as Santha-Vazirani sources [17], Bitcoin beacons [4], and SHELA sources [1]. Crucially, in such versions it is not possible to deterministically extract uniformly random bits.

The execution-leaks model is related to the standard model in secure MPC where the adversary is monolithic, i.e., when a corrupted party P_i is about to be executed, it is the adversary who receives input $(x_1, \ldots x_{i-1}, s_{1,i}, \ldots, s_{i-1,i})$ and determines the outputs $(x_i, s_{i,i+1}, \ldots, s_{i,n})$, and the adversary can see what is sent to corrupted parties as soon as the messages are sent. This is opposed to a model where corrupted party is corrupted independently and only can communicate within the model. In such a model a future corrupted party could not send information back to a previous corrupted party as there is no channel for

this. Such a model is related to the notion of a local adversary, as introduced in [6], though there are technical differences.

1.3 Our Contributions

We obtain both feasibility and impossibility results for randomness extraction in our models. In the context of feasibility, we improve on the naive VSS- and committee-based approach described above, leading to the following results in the sending-leaks and execution-leaks adversarial models. For formal definitions of the models we consider, we refer the reader to Section 2.

Theorem 1 (Feasibility in the sending-leaks model). *There is a zero-error n-party randomness extraction protocol secure against any sending-leaks adversary with corruption threshold t whenever $n \geq 6t + 1$.*

We can improve the theorem above in the execution-leaks model.

Theorem 2 (Feasibility in the execution-leaks model). *There is a zero-error n-party randomness extraction protocol secure against any execution-leaks adversary with corruption threshold t whenever $n \geq 5t$.*

To complement our feasibility results, we also prove an upper bound on the maximum corruption threshold that allows for low-error randomness extraction in our model. This bound also allows us to separate the "random corruptions" regime in YOSO [11], where $t = (1/2 - \delta)n$ random corruptions can be tolerated for any constant $\delta > 0$, from our "worst-case corruptions" regime.

Theorem 3 (Impossibility result). *There is no randomness extraction protocol secure against t corruptions with $n \leq 4t$ parties and bias less than $1/100$ in both the sending-leaks or execution-leaks models.*

For example, combining Theorems 2 and 3 allows us to conclude that the maximal corruption threshold $t^\star$ in the execution-leaks model lies somewhere between $\lfloor n/5 \rfloor$ and $\lceil n/4 \rceil - 1$. Moreover, it follows that $n = 5$ parties are both sufficient and necessary for low-error randomness extraction with $t = 1$ worst-case corruptions in both models.

Extracting Multiple Random Bits. Observe that in order to extract a λ-bit string of unbiased random bits, we can just run our protocols λ times in parallel. This incurs an extra factor of λ in the total communication complexity. We leave it as an interesting open problem to improve on this approach.

Communication complexity and scalability. Our protocols incur communication complexity growing exponentially with n, the number of roles to be executed. Moreover, it is also the case that any protocol in our setting requires time and communication complexity $\Omega(n)$. This raises the question of whether protocols in our setting can scale as the number of users increases. We believe

this to be the case since the number of roles n is detached from the number of users N in the ground set, and it typically holds that $N \gg n$. Nevertheless, improving the concrete efficiency of protocols is very much relevant and we leave it as an interesting future direction to improve on the communication complexity of our protocols.

1.4 Technical Overview

Protocols for Randomness Extraction. We begin by discussing the approach behind the feasibility results in Theorems 1 and 2. Our starting point is an elegant MPC protocol due to Maurer [15] which we modify by taking advantage of the YOSO structure of our models. We present here a sub-optimal version of the protocol for t corruptions and $n = 6t + 2$ parties, and then briefly discuss how we optimize it in the different settings. Divide the set of parties $P_1, \ldots, P_n$ into two consecutive blocks of size $3t + 1$: The *verifiers* $P_1, \ldots, P_{3t+1}$ and the *publishers* $P'_1 = P_{3t+2}, \ldots, P'_{3t+1} = P_{6t+2}$. Intuitively, the protocol proceeds as follows:

1. **Sampling and verification phases:** Do as follows for each set $S \subseteq [3t+1]$ of size $2t + 1$ in parallel:
   (a) Party $P_{i=\min S}$ samples a value x_S uniformly at random from $\{0, 1\}$ and sends it to P_j for all $j \in S \setminus \{i\}$.
   (b) Each such party P_j then forwards the value it received from P_i to all parties $P_{j'}$ with $j' > j$ and $j' \in S$.
   (c) If a party in this process notices an inconsistency between received secret values, it publicly complains about the set S. Else, it sends the consistent value to all publishers $P'_{j''}$ with $j'' \in S$.
2. **Publishing phase:** For every set S as above which did not receive a complaint, each publisher $P'_{j''}$ with $j'' \in S$ publishes the majority of the values it received from verifiers P_j with $j \in S$.

We show that we can obtain an unbiased random bit from this protocol simply by XORing the public values output by all publishers for sets S which did not receive a complaint, even in the stronger sending-leaks setting. The reasons for this are that (i) there exists a set $S^\star$ such that the parties $(P_i, P'_i)_{i \in S^\star}$ are all honest, since $3t + 1 - |S^\star| = t$, and (ii) there is a strict honest majority in all tuples $(P_i)_{i \in S}$ and $(P'_i)_{i \in S}$ for all sets S, since $|S| = 2t + 1$. Roughly speaking, these two properties enforce that an adversary must commit to the final values associated to all sets S *before* the publishing phase, and that at this point the adversary has no knowledge of the value associated to $S^\star$. This ensures that the final XOR is unbiased. In the sending-leaks setting where the adversary learns the secret values sent to corrupted parties as soon as they are sent, we can optimize the protocol above and reduce the number of parties from $6t + 2$ to $6t + 1$, leading to Theorem 1.

In the execution-leaks setting where the adversary only learns the secret values to a corrupted party when it is executed later in the protocol, we optimize

the protocol above first by observing that the publishing phase is wasteful. In fact, we can reduce the number of publishers from $3t+1$ to $2t+1$ and have each verifier send its value to *all* publishers instead. Exploiting the structure of the execution-leaks setting, it follows that the properties detailed above still hold and so the final XOR is still unbiased. This yields a protocol in the execution-leaks setting for t corruptions and $n = 5t + 2$ parties, which is shy of Theorem 2.

The path towards improving on this result in the execution-leaks setting and finally arriving at Theorem 2 is motivated by the following simple (and, as we show, optimal) protocol for $n = 5$ parties and $t = 1$ corruptions, which does not completely fit the steps above because the set of verifiers is too small:

1. Parties P_1 and P_2 sample uniformly random bits x_1 and x_2, respectively, and send them to parties P_3, P_4, and P_5.
2. Parties P_3, P_4, and P_5 publish all secret values they receive.
3. Extract an unbiased bit by XORing the majority of the values attributed to P_1 and P_2, respectively.

As our final optimization, we show that we can modify the general protocol above and reduce the number of verifiers from $3t + 1$ to $3t - 1$, leading to Theorem 2 and thus obtaining this simple protocol as a natural special case. More details can be found in Section 3.

Impossibility Result. We now discuss the approach behind our impossibility result in Theorem 3. For the sake of simplicity, consider a protocol with four parties $P_1, \ldots, P_4$ and one corruption, and assume that a (close to) final output bit is produced if all parties behave honestly. Our proof follows a careful sequential argument where we analyze what would happen if we corrupted parties P_4 through P_1. At a high level, we show that either the behavior of parties $P_1, \ldots, P_{i-1}$ already fully determines the final output of the protocol with high probability, or corrupting P_i allows an adversary to locally control and bias the final output. Care is needed in all stages to ensure the adversary can accurately predict the final output of the protocol locally. We now discuss the main ideas behind each of the four cases:

1. **Corrupt P_4:** In this case, it is easy to see that either the behavior of P_1, P_2, P_3 fully determines the final output with high probability, or there is a decent probability that there are two public values P_4 can publish that would lead to final output 0 and 1, respectively. Therefore, in the latter scenario corrupting P_4 allows us to bias the final output.
2. **Corrupt P_3:** First, we may now assume that the final output is fully determined by P_1, P_2, P_3 with high probability. We use this to show that either corrupting P_3 allows us to bias the final output, or the final output is not only already fully determined by the behavior of P_1 and P_2, *but actually is "independent" of the secret value sent by P_1 to P_3* with high probability. This stronger property will prove useful in the next step where we try to corrupt P_2.

3. **Corrupt P_2:** The goal here is, again, to prove that, assuming it is not useful to bias neither P_3 nor P_4, either we can bias the final output by corrupting P_2 or the final output is fully determined by P_1. However, a main difficulty in this step that is not present elsewhere is that P_2 *does not see the secret value sent from* P_1 *to* P_3. We exploit the stronger statement we proved for P_3 to argue that this is not problematic.
4. **Corrupt P_1:** Finally, assuming that corrupting one of P_2, P_3, P_4 is not useful, we show that P_1 can locally determine the final output based on its public and secret values with high probability. Since we assumed that an honest execution of the protocol yields a nearly unbiased bit, we can corrupt P_1, simulate several runs of the protocol, and always choose the one that leads to, say, final output 0 with high probability.

The above argument shows that such a protocol cannot possibly be secure, which leads to Theorem 3. More details can be found in Section 4.

1.5 Directions for Future Research

Our work leaves open several interesting avenues for future research. We highlight some of them here:

- We have proved, in particular, that the minimum number of parties n required for low-error randomness extraction given a corruption threshold t satisfies $4t+1 \leq n \leq 5t$ in the execution-leaks model and $4t+1 \leq n \leq 6t+1$ in the sending-leaks model. It would be interesting to improve those bounds further towards a complete characterization.
- In this work we did not focus on the efficiency of randomness extraction protocols. Given the practical connections of our models, it would be interesting to design more efficient protocols in this setting. Efficiency could be measured in the number of secret messages and/or the total number of bits sent.
- In line with the previous item, and as already mentioned above, it would be interesting to improve the communication complexity necessary for extracting λ unbiased random bits beyond running our protocols λ times in parallel.
- We considered only static corruptions. However, it also makes sense to consider an active adversary in our model. We leave this as another interesting future modification.

2 Network Models for Randomness Extraction

In this section, we formally define the network models and security notions under which we will be working.

Suppose there are n parties $P_1, \ldots, P_n$. In the first round, party P_1 outputs a public value x_1 and send secret values $s_{1,2}, \ldots, s_{1,n}$ to be received by parties speaking in rounds $i = 2, \ldots, n$, respectively. In the i-th round for $2 \leq i \leq$

n, party P_i outputs a public value $x_i \in \mathcal{X}$ which depends on the previously broadcast public values

$$x^{<i} = (x_1, \ldots, x_{i-1})$$

along with the secret values sent to the party speaking in the i-th round,

$$s^{<i} = (s_{1,i}, \ldots, s_{i-1,i}).$$

Party P_i then sends secret values

$$s^{>i} = (s_{i,i+1}, \ldots, s_{i,n})$$

to be received by parties speaking in rounds $i+1$ through n, respectively. At the end of the protocol, the goal is to *deterministically* extract from the public values $x_1, \ldots, x_n$ a bit that is statistically close to uniform. More precisely, a *randomness extraction protocol* is specified by a tuple $\Pi = (D_1, \ldots, D_n, \mathrm{Ext})$, where $D_1, \ldots, D_n$ are distributions such that

$$(x_i, s^{>i}) \leftarrow D_i(x^{<i}, s^{<i})$$

and $\mathrm{Ext} : \mathcal{X}^n \to \{0,1\}$ is a deterministic function such that the final output of the protocol is given by

$$r = \mathrm{Ext}(x_1, \ldots, x_n).$$

We additionally consider a computationally-unbounded adversary which is allowed to corrupt a subset of parties C of size $|C| \leq t$. We call t the *corruption threshold*. The adversary is taken to be *static*, i.e., the set C is chosen before the start of the protocol. For each party $i \in C$, the adversary is allowed to replace the distribution D_i by an arbitrary malicious distribution M_i of its choice. We study randomness extraction protocols in two natural models depending on the information made available to the adversary when it chooses each M_i, as described next.

2.1 The Sending-Leaks Adversarial Model

In the strongest adversarial model we consider, a *sending-leaks adversary* immediately learns secret values once they are sent. More precisely, for each corrupted party $i \in C$, the adversary may choose the malicious distribution M_i above as a randomized function of the public values $x_1, \ldots, x_{i-1}$ and all values $(s_{j,j'})_{j<i, j' \in C}$. In words, if the adversary corrupts P_i, then it is allowed to see the previously broadcast public values along with all secret values sent by parties 1 through $i-1$ to *all* corrupted parties $j' \in C$, even when $j' > i$. Given a randomness extraction protocol $\Pi = (D_1, \ldots, D_n, \mathrm{Ext})$ and a sending-leaks adversary $\mathcal{A}$, we denote the output of the extractor Ext under the adversarial corruptions imposed by $\mathcal{A}$ by $R(\Pi, \mathcal{A})$. We now present the associated security definition.

Definition 1 (Security in the sending-leaks model). *We say that a randomness extraction protocol Π is* (ε, t)-secure in the sending-leaks model *if for all sending-leaks adversaries $\mathcal{A}$ corrupting at most t parties we have*

$$\left|\Pr[R(\Pi, \mathcal{A}) = 1] - \frac{1}{2}\right| \leq \varepsilon.$$

We say that the protocol is zero-error *if we can take $\varepsilon = 0$.*

2.2 The Execution-Leaks Adversarial Model

We also consider a weaker adversarial model where secret values sent to some corrupted party P_i are only revealed to the adversary once P_i is executed. More precisely, an *execution-leaks adversary* $\mathcal{A}$ chooses the malicious distribution M_i as a randomized function of the public values $x_1, \ldots, x_{i-1}$ and only the secret values $(s_{j,j'})_{1 \leq j < j' \leq i}$. As before, we denote the output of the extractor Ext under the adversarial corruptions imposed by $\mathcal{A}$ by $R(\Pi, \mathcal{A})$, and define security as follows.

Definition 2 (Security in the execution-leaks model). *We say that a randomness extraction protocol Π is* (ε, t)-secure in the execution-leaks model *if for all execution-leaks adversaries $\mathcal{A}$ corrupting at most t parties we have*

$$\left|\Pr[R(\Pi, \mathcal{A}) = 1] - \frac{1}{2}\right| \leq \varepsilon.$$

We say that the protocol is zero-error *if we can take $\varepsilon = 0$.*

3 Zero-Error Randomness Extraction Protocols

We prove our main feasibility results in this section.

3.1 Zero-Error Randomness Extraction in the Sending-Leaks Model

We prove the following feasibility result in the sending-leaks model.

Theorem 4 (Restatement of Theorem 1). *There is a zero-error randomness extraction protocol in the sending-leaks model for corruption threshold t and $n = 6t + 1$ parties.*

We now describe the protocol used to prove Theorem 4, which is a "YOSO-version" of a protocol introduced by Maurer [15]. For the sake of clarity, we first fix a corruption threshold t and $n = 6t + 2$ parties, which we subdivide into consecutive blocks $P_1, P_2, \ldots, P_{3t+1}$, which we call the *verifiers*, and $P'_1, \ldots, P'_{3t+1}$, which we call the *publishers*. We will then show how to optimize this argument so that only $6t + 1$ parties are needed. First, define

$$\mathcal{S} = \{S \subseteq [3t + 1] : |S| = 2t + 1\}.$$

The protocol proceeds as follows:

1. **Sampling phase:** For $i = 1, \ldots, t+1$ and all sets $S \in \mathcal{S}$ such that $\min S = i$, party P_i samples x_S uniformly at random from $\{0,1\}$. We call P_i the *leader* of S. Then, P_i sends x_S to every party P_j such that $j \in S$.
2. For each set $S \in \mathcal{S}$ and $j \in S$, let x_S^j denote the value received by P_j from the leader of S. Then, P_j sends x_S^j to every $P_{j'}$ such that $j' > j$ and $j' \in S$.
3. **Verification phase:** For each set $S \in \mathcal{S}$ and $j' \in S$, let $x_S^{j,j'}$ denote the value received by $P_{j'}$ from P_j for $j' > j$ and $j, j' \in S$. Then, every party $P_{j'}$ checks whether $x_S^{j,j'} = x_S^{j'}$ for all $j \in S$ such that $j < j'$. If this does not hold, then $P_{j'}$ broadcasts the public value $(\text{COMPLAIN}, S)$. Otherwise, the verifier $P_{j'}$ sends $x_S^{j'}$ to all publishers $P'_{j''}$ such that $j'' \in S$.
4. **Publishing phase:** For each $S \in \mathcal{S}$ which did not receive a complaint and $j', j'' \in S$, let $y_S^{j',j''}$ denote the value received by the publisher $P'_{j''}$ from the verifier $P_{j'}$. Then, $P'_{j''}$ broadcasts $(S, s_S^{j''})$, where

$$s_S^{j''} = \mathsf{maj}((y_S^{j',j''})_{j' \in S})$$

and maj denotes the majority function with ties broken to 0.

Given the values broadcast by the protocol above, our randomness extractor behaves as follows: First, for all $S \in \mathcal{S}$ which did not receive a complaint, set

$$m_S = \mathsf{maj}((s_S^j)_{j \in S}).$$

For all sets $S \in \mathcal{S}$ which received a complaint, set $m_S = 0$. Then, the output of the extractor is

$$\bigoplus_{S \in \mathcal{S}} m_S.$$

The following statement holds.

Proposition 1. *The value $\bigoplus_{S \in \mathcal{S}} m_S$ is uniformly random whenever at most t out of $n = 6t + 2$ parties are corrupted.*

Proof. Observe that there is a set $S^\star \in \mathcal{S}$ such that all verifiers P_j and publishers P'_j with $j \in S^\star$ are honest. In particular, this means that the value $x_{S^\star}$ sampled by the leader of $S^\star$ is uniformly random, that $S^\star$ does not receive a complaint, and that $m_{S^\star} = x_{S^\star}$. Moreover, since all associated publishers $(P'_j)_{j \in S^\star}$ are honest, the value of $x_{S^\star}$ is not leaked to the adversary before the publishing phase. It remains to argue that the values m_S for $S \neq S^\star$ are independent of $x_{S^\star}$, which concludes the proof.

Consider any set $S \neq S^\star$ in $\mathcal{S}$. Note that whether S receives a complaint or not is independent of the value of $x_{S^\star}$ since corrupted parties only learn this value later in the publishing phase. Therefore, it suffices to consider the case where S did not receive a complaint. If this holds, it must be the case that all honest parties P_j for $j \in S$ received the same value x'_S from the leader of S, which is independent of $x_{S^\star}$. Otherwise, if honest parties P_{j_1} and P_{j_2} received different values and $j_1 < j_2$, then P_{j_2} would fail the check and broadcast a complaint

during the verification phase. Since a strict majority of verifiers in $(P_j)_{j\in S}$ is honest, it follows that all honest publishers $P'_{j'}$ for $j' \in S$ will broadcast $s_S^{j'} = x'_S$. Therefore, we have $m_S = x'_S$ since a strict majority of publishers in $(P'_{j'})_{j'\in S}$ is honest, which yields the desired claim. □

In order to obtain Theorem 4 from Proposition 1, it remains to describe how to modify the protocol in order to reduce the number of parties from $6t + 2$ to $6t + 1$. This can be accomplished by merging parties P_{3t+1} and P'_{3t+1} into one party. Since the publishing phase of the protocol is insensitive to the broadcast order and to the fact that each publisher P'_j may share a state with the verifier P_j, the correctness of the protocol still holds.

3.2 Improved Zero-Error Randomness Extraction in the Execution-Leaks Model

The protocol described in Section 3.1 is secure in the strong adversarial model where values sent to corrupted parties are immediately displayed to the adversary. However, we can also consider the natural execution-leaks model where parties only learn their values when they are executed. Our impossibility bound also holds in this model, meaning that $n \geq 4t + 1$ parties are necessary in this model as well. Therefore, it is natural to wonder whether a bound better than $n \leq 6t + 1$ is achievable in this case. It turns out that in this model we can optimize our protocol above and prove the following result.

Theorem 5 (Restatement of Theorem 2). *There is a zero-error randomness extraction protocol in the execution-leaks model for corruption threshold t and $n = 5t$ parties.*

Our starting point towards proving Theorem 5 is a protocol for $n = 5t + 2$ parties.

Proposition 2. *There is a zero-error randomness extraction protocol in the execution-leaks model for corruption threshold t and $n = 5t + 2$ parties.*

Proof. The protocol proceeds as in Section 3.1 except for the following differences:

- There are $2t + 1$ publishers $P'_1, \ldots, P'_{2t+1}$ instead of $3t + 1$;
- In the verification phase (Step 3), the verifier $P_{j'}$ sends $x_S^{j'}$ to *all* publishers $P'_1, \ldots, P'_{2t+1}$;
- In the publishing phase (Step 4), one takes $j' \in S$ and $j'' \in [2t+1]$ arbitrary.

The correctness of the modified protocol follows as in the proof of Proposition 1, except that we only use the fact that there exists a set $S^\star \in \mathcal{S}$ such that all verifiers $(P_j)_{j\in S^\star}$ are honest and that a strict majority of the verifiers is honest. □

We now show how we can improve the protocol above so that $n = 5t$ parties are enough, yielding Theorem 5.

Proof (Theorem 5). The protocol proceeds as in the proof of Proposition 2 except for the following differences:

- There are $3t-1$ verifiers $P_1, \ldots, P_{3t-1}$ instead of $3t+1$;
- The family $\mathcal{S}$ is now defined as

$$\mathcal{S} = \{S \subseteq [3t-1] : |S| = 2t-1\}.$$

The correctness of the protocol follows by case analysis:

1. **If t verifiers are corrupted:** This implies that all publishers $P'_1, \ldots, P'_{2t+1}$ are honest. Moreover, there is a set $S^\star \in \mathcal{S}$ of fully honest verifiers $(P_i)_{i \in S^\star}$. Since the output of the extractor is fully determined after all verifiers speak and the adversary does not observe $x_{S^\star}$, the output of the extractor is uniformly random.
2. **If at most $t-1$ verifiers are corrupted:** In this case all sets $S \in \mathcal{S}$ have a strict majority of honest parties (since $2t-1 > 2(t-1)$), a strict majority of publishers is honest, and there is a set $S^\star \in \mathcal{S}$ containing only honest parties. Therefore, the argument from the proof of Proposition 2 goes through and shows that the output of the extractor is uniformly random. □

Combining Theorems 5 and 6 leads to the following exact characterization of the round complexity of randomness extraction for $t = 1$ corruptions in both models.

Corollary 1. *A total of $n = 5$ parties are both sufficient and necessary for low-error randomness extraction with $t = 1$ corruptions (in both the sending-leaks and execution-leaks models).*

4 Low-Error Randomness Extraction is Impossible with $n/4$ Corruptions

We prove our impossibility result in this section.

Theorem 6 (Restatement of Theorem 3). *There is no $(\varepsilon = 0.01, t)$-secure randomness extraction protocol in both the sending-leaks and execution-leaks models for n parties and corruption threshold $t \geq n/4$.*

Proof. At a high level, we prove this result by dividing the parties into four consecutive blocks $B_1, \ldots, B_4$ and then sequentially arguing that either the behavior of $B_1, \ldots, B_{i-1}$ already fully determines the final output of the protocol with high probability, or corrupting B_i allows an adversary to locally control and bias the final output by consistently resampling B_i's public value and secret values. Care is needed in all stages to ensure the adversary can accurately predict the final output of the protocol locally.

Fix some randomness extraction protocol $\Pi = (D_1, \ldots, D_n, f)$ with corresponding public values $X_1, \ldots, X_n$. Let $f(X_1, \ldots, X_n)$ denote the output of the YOSO extraction protocol, where f is a deterministic function and $t = \lceil n/4 \rceil$

parties may be corrupted by an execution-leaks adversary. With a contradiction in view, suppose that

$$|\Pr[f(X_1,\ldots,X_n)=1]-\Pr[f(X_1,\ldots,X_n)=0]|\leq\varepsilon=0.01, \tag{1}$$

where the probability is taken over the randomness of the protocol. Partition the set of parties $[n]$ into four consecutive blocks B_1, B_2, B_3, B_4 each containing at most $\lceil n/4\rceil$ parties. Note that the adversary is able to corrupt all parties in one of these blocks. Let Q_i denote the string of public values output by the block B_i and $S_{i\to j}$ denote the set of secret values sent by parties of block B_i to parties of block B_j for $i<j$.

We begin by considering the case where the adversary fully corrupts the block B_4. Sample $(q_1,q_2,q_3)\leftarrow(Q_1,Q_2,Q_3)$. By our assumption, it must be the case that

$$\Pr\left[\exists q_4^{(0)},q_4^{(1)}:f\left(q_1,q_2,q_3,q_4^{(b)}\right)=b,b\in\{0,1\}\right]\leq\varepsilon, \tag{2}$$

where the probability is taken over the sampling above. In other words, the last block of parties must have little control over the output of the extractor. To see this, note that if (2) did not hold then we could simply have the adversary make block B_4 select $q_4^{(0)}$ as the public value (whenever such choices exist) so that (1) is not satisfied.

We move to the case where the adversary fully corrupts B_3. Suppose blocks B_1 and B_2 output public and secret value sets $(q_1,s_{1\to3},q_2,s_{2\to3})$ according to the joint distribution $(Q_1,S_{1\to3},Q_2,S_{2\to3})$, which the adversary sees. Set $s_{1\to3}^{(1)}=s_{1\to3}$ and sample $s_{1\to3}^{(2)}$ according to the distribution

$$(S_{1\to3}|Q_1=q_1,Q_2=q_2,S_{2\to3}=s_{2\to3}).$$

Then, sample

$$q_3^{(i)}\leftarrow(Q_3|Q_1=q_1,Q_2=q_2,S_{1\to3}=s_{1\to3}^{(i)},S_{2\to3}=s_{2\to3})$$

for $i=1,2$. Handling this extra sample from $S_{1\to3}$ will be crucial for our argument later on in the case where we corrupt B_2. We define the set of *good* tuples

$$\mathcal{G}_3=\{(q_1',q_2',q_3'):f(q_1',q_2',q_3',\cdot)\text{ is constant}\}$$

and claim that

$$\Pr\left[(q_1,q_2,q_3^{(1)})\in\mathcal{G}_3,(q_1,q_2,q_3^{(2)})\in\mathcal{G}_3,f(q_1,q_2,q_3^{(1)},\cdot)\equiv f(q_1,q_2,q_3^{(2)},\cdot)\right]\geq1-3\varepsilon, \tag{3}$$

where the probability is taken over the sampling procedures described above. To see this, first note that both $q_3^{(1)}$ and $q_3^{(2)}$ are distributed like a correct public value of B_3 in the protocol, i.e., the tuples $(q_1,q_2,q_3^{(1)})$ and $(q_1,q_2,q_3^{(2)})$ are both distributed according to the joint distribution (Q_1,Q_2,Q_3). Therefore, invoking (2) and the union bound shows that

$$\Pr[(q_1,q_2,q_3^{(1)})\in\mathcal{G}_3,(q_1,q_2,q_3^{(2)})\in\mathcal{G}_3]\geq1-2\varepsilon. \tag{4}$$

Moreover, it must be the case that

$$\Pr\left[(q_1, q_2, q_3^{(1)}) \in \mathcal{G}_3, (q_1, q_2, q_3^{(2)}) \in \mathcal{G}_3, f(q_1, q_2, q_3^{(1)}, \cdot) \not\equiv f(q_1, q_2, q_3^{(2)}, \cdot)\right] \leq \varepsilon, \tag{5}$$

since otherwise the adversary could sample $q_3^{(1)}$ and $q_3^{(2)}$ as above, each of which would fully determine the output of the extractor with probability larger than ε, and so bias the extractor appropriately, thus implying that (1) is false. Combining (5) and (4) yields (3), as desired.

Continuing this trend, suppose now that the adversary fully corrupts B_2 and that block B_1 outputs public and secret values $(q_1, s_{1\to 2}, s_{1\to 3})$. Note, however, that the adversary only has access to q_1 and $s_{1\to 2}$. We will exploit (3) to argue that the adversary can still adequately simulate Q_3 and Q_4 for a given choice of its public value Q_2 and predict the output of f with decent probability by simulating the values from B_1 to B_3 locally. First, independently sample pairs

$$(q_2^{(i)}, s_{2\to 3}^{(i)}) \leftarrow (Q_2, S_{2\to 3} | Q_1 = q_1, S_{1\to 2} = s_{1\to 2})$$

for $i = 1, 2$. Then, independently sample *simulated* secret value sets

$$\widetilde{s}_{1\to 3}^{(i)} \leftarrow (S_{1\to 3} | Q_1 = q_1, Q_2 = q_2^{(i)}, S_{2\to 3} = s_{2\to 3}^{(i)})$$

and simulated public values

$$\widetilde{q}_3^{(i)} \leftarrow (Q_3 | Q_1 = q_1, Q_2 = q_2^{(i)}, S_{1\to 3} = s_{1\to 3}^{(i)}, S_{2\to 3} = s_{2\to 3}^{(i)})$$

for $i = 1, 2$. Denote by

$$f(q_1', q_2', q_3'; m_{1\to 3}', m_{2\to 3}')$$

the (possibly randomized) output of $f(Q_1, Q_2, Q_3, Q_4)$ conditioned on the joint event

$$(Q_1 = q_1', Q_2 = q_2', Q_3 = q_3', S_{1\to 3} = m_{1\to 3}', S_{2\to 3} = m_{2\to 3}').$$

Then, we define the set of good tuples

$$\mathcal{G}_2 = \{(q_1', q_2', q_3', m_{1\to 3}', m_{2\to 3}') : f(q_1', q_2', q_3'; m_{1\to 3}', m_{2\to 3}') \text{ is constant}\}.$$

Suppose that $q_3^{(i)}$ is the true public value of B_3 when the corrupted block B_2 outputs $q_2^{(i)}$ and sends values $s_{2\to 3}^{(i)}$. For convenience, we set

$$\begin{aligned}
\mathbf{v}^{(i)} &= (q_1, q_2^{(i)}, q_3^{(i)}, s_{1\to 3}, s_{2\to 3}^{(i)}), \\
\widetilde{\mathbf{v}}^{(i)} &= (q_1, q_2^{(i)}, \widetilde{q}_3^{(i)}; \widetilde{s}_{1\to 3}, s_{2\to 3}^{(i)}), \\
r^{(i)} &= f(\mathbf{v}^{(i)}), \\
\widetilde{r}^{(i)} &= f(\widetilde{\mathbf{v}}^{(i)}).
\end{aligned}$$

Using this notation, combining (3) with a union bound over $i = 1, 2$ yields

$$\Pr[\forall i \in \{1, 2\} : \mathbf{v}^{(i)} \in \mathcal{G}_2, r^{(i)} \equiv \widetilde{r}^{(i)}] \geq 1 - 6\varepsilon. \tag{6}$$

In words, the adversary can predict the final output of the protocol if he decides on $(q_2^{(i)}, s_{2\to 3}^{(i)})$ with high probability by locally computing $\widetilde{r}^{(i)}$ for $i = 1, 2$. Moreover, similarly to previous cases, it must be that

$$\Pr[\forall i \in \{1,2\} : \mathbf{v}^{(i)} \in \mathcal{G}_2, r^{(i)} \equiv \widetilde{r}^{(i)}, \widetilde{r}^{(1)} \not\equiv \widetilde{r}^{(2)}] \leq \varepsilon. \tag{7}$$

In fact, if this did not hold, then the adversary could locally sample the tuples $(q_2^{(i)}, s_{2\to 3}^{(i)}, \widetilde{q}_3^{(i)}, \widetilde{s}_{1\to 3}^{(i)})$, compute $\widetilde{r}^{(i)}$, and then choose i such that $\widetilde{r}^{(i)} = 0$ and behave accordingly, thus biasing the output to 0 by more than ε. Therefore, combining (6) and (7) implies that

$$\Pr[\forall i \in \{1,2\} : \mathbf{v}^{(i)} \in \mathcal{G}_2, r^{(i)} \equiv \widetilde{r}^{(i)}, \widetilde{r}^{(1)} \equiv \widetilde{r}^{(2)}] \geq 1 - 7\varepsilon. \tag{8}$$

This means that we can predict the final output of f with high probability given only $(q_1, s_{1\to 2})$.

Finally, we consider the case where the adversary corrupts the first block B_1. Consider two independent samples $(q_1^{(1)}, s_1^{(1)})$ and $(q_1^{(2)}, s_1^{(2)})$ according to the joint distribution (Q_1, S_1), where S_1 denotes all values sent by parties in B_1. Then, if (1) holds it must be the case that the output of f differs between the runs of the protocol beginning with $(q_1^{(1)}, s_1^{(1)})$ and $(q_1^{(2)}, s_1^{(2)})$ with some probability $p > 1/2 - \varepsilon^2$. By (2), (3), and (8), the output of f on the respective runs of the protocol is fully determined by knowledge of $(q_1^{(1)}, s_1^{(1)})$ and $(q_1^{(2)}, s_1^{(2)})$ with probability at least $1 - 2 \cdot 7\varepsilon = 1 - 14\varepsilon$. This implies that with probability at least

$$p - 14\varepsilon > 1/2 - 14\varepsilon - \varepsilon^2 \geq \varepsilon$$

the adversary can completely bias the output of f by choosing the run which leads to output 0, which contradicts (1). Since all steps above only involve resampling the distributions of honest parties, this strategy can be implemented by a execution-leaks adversary. □

We also show that we cannot hope to prove a better upper bound using the ideas above, as formalized in the following theorem.

Theorem 7. *There exists a zero-error randomness extraction protocol with $n = 4t + 1$ parties in both the sending-leaks and execution-leaks models when the adversary is only allowed to corrupt at most t* consecutive *parties.*

Proof. Fix a corruption threshold t and $n \geq 4t + 1$ parties. Consider a protocol where parties $i \in [t + 1]$ (the generators) publish 0 and send a random bit b_i to all parties in $\{2t + 1, \ldots, n\}$. Parties $i \in \{t + 2, \ldots, 2t\}$ are silent (they do not publish public values nor do they send any secret values). Let b_i^j denote the bit received by the j-th party from the i-th party. Then, parties $j \in \{2t + 1, \ldots, n\}$ (the publishers) each output $b^j = \bigoplus_{i=1}^{t+1} b_i^j$. Finally, the extractor computes

$$\mathsf{maj}(b^{2t+2}, \ldots, b^n).$$

Suppose there is an adversary which corrupts a block of t consecutive parties. Note that the adversary cannot corrupt generators and publishers simultaneously, since there are $t-1$ silent parties in between. Moreover, the adversary gains nothing by corrupting silent parties, since their public values and values may just be ignored. It remains to consider two cases:

- The adversary corrupts at most t generators: Without loss of generality, suppose the first generator is honest. Then, it holds that the adversarial choice of the bits b_i^j for $i \in \{2, \ldots, t+1\}$ is independent of b_1, which is uniformly random, and $b_1^j = b_1$ for all publishers j. It follows that the majority is uniformly random as well.
- The adversary corrupts at most t publishers: Since there are $2t+1$ publishers and all generators are honest, the majority coincides with $\bigoplus_{i=1}^{t+1} b_i$, which is uniformly random.

We conclude that the protocol outputs a uniformly random bit. □

Acknowledgments. JBN was partially funded by The Concordium Foundation; The Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE); The Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM). JR was supported in part by the NSF grants CCF-1814603 and CCF-2107347 and by the following grants of Vipul Goyal: the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award. MO was supported by MOE2019-T2-1-145 Foundations of quantum-safe cryptography.

JR thanks Chen-Da Liu Zhang, Elisaweta Masserova, and Justin Raizes for insightful discussions.

References

1. Aggarwal, D., Obremski, M., Ribeiro, J., Siniscalchi, L., Visconti, I.: How to extract useful randomness from unreliable sources. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 343–372. Springer International Publishing, Cham (2020)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing. pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213
3. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) Theory of Cryptography - 18th International Conference, TCC 2020. pp. 260–290. Springer (2020). https://doi.org/10.1007/978-3-030-64375-1_10
4. Bentov, I., Gabizon, A., Zuckerman, D.: Bitcoin beacon. CoRR **abs/1605.04559** (2016), http://arxiv.org/abs/1605.04559

5. Campanelli, M., David, B., Khoshakhlagh, H., Kristensen, A.K., Nielsen, J.B.: Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. IACR Cryptol. ePrint Arch. (2021), https://eprint.iacr.org/2021/1423
6. Canetti, R., Vald, M.: Universally composable security with local adversaries. In: Visconti, I., Prisco, R.D. (eds.) Security and Cryptography for Networks - 8th International Conference. pp. 281–301. Springer (2012). https://doi.org/10.1007/978-3-642-32928-9_16
7. Cascudo, I., David, B.: ALBATROSS: Publicly AttestabLe BATched Randomness Based On Secret Sharing. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020. pp. 311–341. Springer (2020). https://doi.org/10.1007/978-3-030-64840-4_11
8. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing. pp. 11–19. ACM (1988). https://doi.org/10.1145/62212.62214
9. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid MPC: Secure multiparty computation with dynamic participants. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021. pp. 94–123. Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_4
10. Das, S., Krishnan, V., Isaac, I., Ren, L.: SPURT: Scalable distributed randomness beacon with transparent setup. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 1380–1395. IEEE Computer Society, Los Alamitos, CA, USA (May 2022). https://doi.org/10.1109/SP46214.2022.00080
11. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You Only Speak Once - secure MPC with stateless ephemeral roles. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021. pp. 64–93. Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_3
12. Gentry, C., Halevi, S., Magri, B., Nielsen, J.B., Yakoubov, S.: Random-index PIR and applications. In: Nissim, K., Waters, B. (eds.) Theory of Cryptography - 19th International Conference, TCC 2021. pp. 32–61. Springer (2021). https://doi.org/10.1007/978-3-030-90456-2_2
13. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: Coppersmith, D. (ed.) Advances in Cryptology - CRYPTO '95. pp. 339–352. Springer (1995). https://doi.org/10.1007/3-540-44750-4_27
14. Hirt, M., Maurer, U.M.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000). https://doi.org/10.1007/s001459910003
15. Maurer, U.: Secure multi-party computation made simple. Discrete Applied Mathematics **154**(2), 370–381 (2006). https://doi.org/10.1016/j.dam.2005.03.020
16. Micali, S.: ALGORAND: the efficient and democratic ledger. CoRR **abs/1607.01341** (2016), http://arxiv.org/abs/1607.01341
17. Santha, M., Vazirani, U.V.: Generating quasi-random sequences from slightly-random sources (extended abstract). In: 25th Annual Symposium on Foundations of Computer Science. pp. 434–440. IEEE Computer Society (1984). https://doi.org/10.1109/SFCS.1984.715945