Improved YOSO Randomness Generation with Worst-Case Corruptions

Chen-Da Liu-Zhang[1], Elisaweta Masserova[2], João Ribeiro[3], Pratik Soni[4], and Sri AravindaKrishnan Thyagarajan[5]

[1] Luzern University of Applied Sciences and Arts & Web3 Foundation, Zug, Switzerland
chen-da.liuzhang@hslu.ch
[2] Carnegie Mellon University, Pittsburgh, PA, USA
elisawem@andrew.cmu.edu
[3] NOVA LINCS & NOVA School of Science and Technology, Caparica, Portugal
joao.ribeiro@fct.unl.pt
[4] University of Utah, Salt Lake City, UT, USA
psoni@cs.utah.edu
[5] NTT Research, Sunnyvale, CA, USA
t.srikrishnan@gmail.com

Abstract. We study the problem of generating public unbiased randomness in a distributed manner within the recent You Only Speak Once (YOSO) framework for stateless multiparty computation, introduced by Gentry et al. in CRYPTO 2021. Such protocols are resilient to adaptive denial-of-service attacks and are, by their stateless nature, especially attractive in permissionless environments. While most works in the YOSO setting focus on independent random corruptions, we consider YOSO protocols with worst-case corruptions, a model introduced by Nielsen et al. in CRYPTO 2022.

Prior work on YOSO public randomness generation with worst-case corruptions designed information-theoretic protocols for t corruptions with either $n = 6t + 1$ or $n = 5t$ roles, depending on the adversarial network model. However, a major drawback of these protocols is that their communication and computational complexities scale exponentially with t. In this work, we complement prior inefficient results by presenting and analyzing simple and efficient protocols for YOSO public randomness generation secure against worst-case corruptions in the computational setting. Our first protocol is based on publicly verifiable secret sharing and uses $n = 3t+2$ roles. Since this first protocol requires setup and somewhat heavy cryptographic machinery, we also provide a second lighter protocol based on ElGamal commitments and verifiable secret sharing which uses $n = 5t + 4$ or $n = 4t + 4$ roles depending on the underlying network model. We demonstrate the practicality of our second protocol by showing experimental evaluations, significantly improving over prior proposed solutions for worst-case corruptions, especially in terms of transmitted data size.

1 Introduction

Public randomness is a fundamental component of numerous financial and security protocols [15, 11]. Randomness usage is ubiquitous: From establishing fairness in the green card lottery, to assessing risk via Monte Carlo simulations, to generating the public parameters for the cryptographic protocols [1, 12]. In the past, public randomness was typically obtained via trusted third parties. However, with the emergence of blockchains and web3, there has been an increased effort to decentralize economic activities, and as a consequence, to decentralize public randomness generation as well [5, 18, 6, 10, 4].

A protocol for such distributed public randomness allows multiple mutually distrusting parties, each with their own source of randomness, to generate and agree on a public random value. However, designing a secure protocol which provides such a functionality is a notoriously hard task. Indeed, the cryptographic community put significant effort into designing distributed randomness generation protocols [5, 6, 18, 10, 4], as well as improving functionalities such as verifiable delay functions [3] and time-lock puzzles [19], which oftentimes serve as building blocks in such protocols.

Traditionally, those protocols consider *static* adversaries, where security is guaranteed as long as the adversary decides which parties to corrupt prior to the start of the execution. However, such an assumption seems unjustified, especially for protocols that run over long periods of time. A far more realistic setting would allow the adversary to corrupt parties *dynamically* during the course of the execution. Unfortunately, protocols in this *adaptive* setting tend to be costly. In the context of general multi-party computation, a novel approach to achieving adaptive security (in an arguably more efficient manner) has been recently proposed in the *You-Only-Speak-Once* (YOSO) line of work, introduced by Gentry, Halevi, Krawczyk, Magri, Nielsen, Rabin, and Yakoubov [7]. Intuitively, protocols in the YOSO setting consider the notion of stateless ephemeral *roles*, where at a single point in time a small committee of such roles is required to perform certain actions, and produce a public output, along with messages to be sent to future roles. Roles are assigned to physical machines via a "role-assignment" functionality in the beginning of each round, in a way that makes it hard for the adversary to predict which physical machines will be participating as roles in a given committee. As roles are allowed to send only a single message (i.e., speak only once), and are torn down after the execution, adaptively corrupting a machine which executed a certain role in the past does not help the adversary. Due to these observations, assuming that the adversary can only corrupt a fraction of (a large total number of) physical machines, the protocols designed in the YOSO setting typically rely on the fact that the adversary's best option is to corrupt machines *at random*.

However, this assumption is viable only if role-assignment (which is typically separated from the multi-party protocol computing the function of interest) is truly secure. This makes role-assignment protocols hard to design, and the currently known constructions compromise either in terms of efficiency [8] or in terms of the supported corruption threshold [2]. In order to reduce trust in role-

assignment, Nielsen, Ribeiro, and Obremski (NRO in the following) recently introduced a model for YOSO with *worst-case corruptions* [13], a model we dub $\text{YOSO}^{\mathsf{WCC}}$. In this model, prior to the start of the protocol, the adversary can choose any up to t roles to corrupt. The $\text{YOSO}^{\mathsf{WCC}}$ model is tailored to the randomness generation setting, and the authors introduce two information-theoretic protocols which are secure given worst-case corruption of roles. Unfortunately, these protocols incur exponential communication- and computation complexities, which motivates us to ask the following question:

Can we design efficient distributed randomness generation protocols in the model of YOSO with worst-case corruptions?

As it is trivially possible to adapt known stateful randomness generation protocols to the $\text{YOSO}^{\mathsf{WCC}}$ setting at the cost of having a very low adversarial threshold (see Section 1.3 for details), we further refine the question as follows:

Can we design efficient distributed randomness generation protocols in the model of YOSO with worst-case corruptions while optimizing the required number of roles?

1.1 Our Contributions

In this work, we answer the question above positively. As in NRO, we distinguish between two different adversarial models, the *sending-leaks* and *execution-leaks* models. Intuitively, in the execution-leaks model the adversary only obtain messages addressed to corrupted parties upon their execution. In the stronger sending-leaks model, the adversary obtains the messages addressed to corrupted parties immediately upon the sender sending the message. We design two randomness generation protocols in the sending-leaks model, along with an optimized version for the execution-leaks scenario, and prove these protocols secure. Our protocols are in the computational setting, meaning that the adversary we consider is computationally bounded.

In our first construction, we build upon a non-interactive *publicly verifiable secret sharing* (PVSS) [17], which allows a dealer to share a secret in a single round among a set of parties (a subset of which can be corrupt) in a way that lets anyone to verify that the dealer behaved correctly. Our PVSS-based randomness generation protocol requires $3t+2$ roles, and has communication complexity that grows quadratically in the number of parties. While this construction requires setup and somewhat heavy cryptographic machinery in the form of simulation-extractable non-interactive zero-knowledge proofs, in our second construction we do not require setup and rely only on the usage of ElGamal commitments. In this construction, we build upon *verifiable secret sharing* (VSS) [14], which is similar to PVSS, but requires more rounds. This protocol has a communication complexity that is quadratic in the number of roles, and requires $5t + 4$ roles in the sending leaks model, or $4t + 4$ in the execution leaks model.

We implement our VSS-based construction and compare it to our implementation of the NRO scheme. Our evaluation shows that our protocol is not only

asymptotically, but also concretely efficient, and we outperform NRO for values as small as $t = 6$ (for running time), and $t = 3$ (for size of the transmitted data).

In the following, we first briefly outline our model, and then provide an overview of the main techniques and ideas used in our work.

1.2 Our Model and Security Goal

We now briefly outline the YOSO^{WCC} model we work in, following the communication model description of NRO [13]. We distinguish between stateless "roles" and physical machines which may run for a long time and retain state. Note that in the following, we use the terms "role" and "party" interchangeably. We consider n parties $P_1, \ldots, P_n$, which are executed one after the other. We assume that each party has its own internal source of randomness. We consider a computationally bounded adversary which is allowed to corrupt any t out of n parties before the protocol starts. Upon its execution, P_i can publicly broadcast a value x_i and send secret values $s_{i,j}$ to each "future" party P_j, i.e., any P_j such that $j > i$. We consider the following two adversarial network settings:

- In the *sending-leaks* model an adversary obtains a message $s_{i,j}$ sent by an honest P_i to a corrupt P_j as soon as P_i sent it. We call the corresponding adversary the *sending-leaks* adversary.
- In the *execution-leaks* model an adversary obtains a message $s_{i,j}$ sent by an honest P_i to a corrupt P_j only once P_j is activated. We call the corresponding adversary the *execution-leaks* adversary.

In this model, our goal is the following: After the execution of all parties is complete, anyone (not just physical machines which acted as roles $P_1, \cdots, P_n$) can obtain unbiased public randomness by applying a publicly known extraction function to the values $(x_1, \ldots, x_n)$. See Figure 1 for a visual representation of this process.

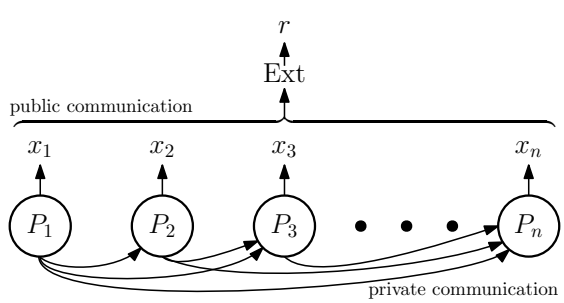

Fig. 1: Communication model from [13, Figure 1]. Parties P_i speak one after the other, send secrets to future parties P_j for $j > i$, and publish public values, which are available to all parties.

More formally, let λ denote a security parameter. Consider an interaction of an adversary A with the honest parties in the randomness generation protocol and let $\mathsf{OUT}(A)$ denote the coin output of this protocol with adversary A. Let $L(\lambda)$ denote the length of this output. Let D be a distinguisher. Consider the following experiment (for protocols which assume trusted setup, this setup is generated by the challenger):

1. $b \xleftarrow{\$} \{0,1\}$.
2. $r \xleftarrow{\$} \{0,1\}^{L(\lambda)}$.
3. If $b = 0$, set $\mathsf{coin} \leftarrow \mathsf{OUT}(A)$. Otherwise, set $\mathsf{coin} \leftarrow r$.
4. $b' \leftarrow D(\mathsf{coin})$.

Definition 1 (Computationally secure YOSO^{WCC} randomness generation). *A YOSO^{WCC} randomness generation protocol with n parties is* (t,n)-computationally secure in the sending-leaks (resp. execution-leaks) model *if for all PPT sending-leaks (resp. execution-leaks) adversaries A that corrupt t out of n parties and all PPT distinguishers D in the above security game it holds that*

$$\left|\Pr[b = b'] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

1.3 Our Techniques

First of all, note that, as pointed out by NRO, any stateful r-round multiparty computation protocol which is secure against t out of n corruptions can be ported to the YOSO^{WCC} setting as follows: Use r roles $P_{i,r}$ to implement the behavior of each participant P_i of the stateful protocol over r rounds. Role $P_{i,k}$ mimics the behavior of P_i in round k of the stateful protocol, with the caveat that it additionally sends its state to the future role $P_{i,k+1}$. Unfortunately, this approach is costly in terms of the required number of roles: It requires $n \cdot r$ roles, while tolerating only t corrupted parties.

To address this issue, we design randomness generation protocols which are tailored to the YOSO^{WCC} setting. For simplicity, say we wish to generate only a single random bit r.

First idea. Our first idea is the following: As each party has its own source of randomness, we could set $n = t + 1$ and simply xor all values r_i, where r_i is the random bit generated by P_i: $r = \bigoplus_{i \in [t+1]} r_i$. As at least one party out of $t + 1$ is honest, the xor should result in an unbiased bit. However, we need to be careful: We must not let a corrupt party see the values of the honest parties before supplying its own r_i. Thus, intuitively, we have to make each party *commit* to the randomness it wishes to contribute prior to revealing the values of other parties. This approach requires a party to speak two times: Once when committing to a value, and once when opening it. This can be naively achieved by using two roles to implement P_i, and having the first role privately send its state to its counterpart.

Surprisingly, this approach still does not achieve what we want: As corrupt parties can refuse to open the committed values, in our protocol we must specify

how to proceed in such case. We can either choose to ignore each such party P_i, thereby making their contribution equal to $r_i = 0$ (first case in the following), or set $r_i = 1$ (second case). In both cases, a corrupt P_{t+1} can bias the outcome of the final xor by committing to $r_{t+1} = 1$ in the first case ($r_{t+1} = 0$ in the second case), and then adaptively deciding whether to open the value or not during the execution of its second role, thereby setting the result r to the value of its choice. As the second role of P_{t+1} is the last party speaking, all values supplied by the honest parties are known upon its execution.

Utilizing PVSS. We address the issue above by ensuring that the coin output is fixed *prior* to the reveal phase. We begin by considering a setting with trusted setup. In this case, we can rely on a (t, n)-*publicly verifiable secret sharing* (PVSS) protocol. Using such a scheme, a dealer can secret share its secret among n parties in a way that any $t + 1$ parties can reconstruct the secret, but any t (potentially corrupted) parties have no information about the secret. Moreover, public verifiability ensures that anyone (even non-recipients) can verify that the dealer sharing has been done correctly, i.e., there exists a *unique* secret which can be later reconstructed by any set of $t + 1$ recipient parties.

Intuitively, this fixes the secret at the end of the commit/sharing phase, and if the adversary corrupts at most t parties, it does not learn any information about the secret. If we ensure that the secret reconstruction starts only after the sharing phase of *all* secrets is complete, the adversary can no longer bias the outcome. However, there is one caveat: As anyone must be able to verify that the sharing was done correctly, the dealer cannot send the shares to the parties via *private* communication. Instead, the dealer publishes encryptions of the shares of the parties with respect to their corresponding public keys. In a scenario such as ours, where we run not only one, but *multiple* PVSS, publicly revealing encryptions of the shares makes PVSS susceptible to malleability attacks. To prevent such attacks from adversarial dealers, we make use of a PVSS protocol with appropriate non-malleability properties. Such properties can be achieved, for example, via simulation-extractable non-interactive zero-knowledge proofs [9].

If we use a $(t, 2t + 1)$-PVSS protocol, the above protocol requires only $3t + 2$ roles in total: $t + 1$ dealers and $2t + 1$ parties who obtain the secret shares. This protocol allows us to achieve the following result:

Theorem 1 (informal). *Assuming public key encryption and simulation-extractable NIZKs, there exists a computationally secure randomness generation protocol with $3t + 2$ roles in the sending-leaks model, where t is the number of corruptions.*

We give a formal description of our PVSS-based construction in Section 2.

Removing Trusted Setup. While the protocol above enjoys good efficiency properties and requires only a small number of parties, it relies on somewhat heavy cryptographic assumptions and a trusted setup. In our second and main construction we address these limitations.

Our idea is to utilize *verifiable secret sharing* (VSS), which is similar to PVSS, except that it does not provide public verifiability. Instead, we only have

the so-called "strong commitment" property, which states that the shares of the honest parties define a secret (which could be $\perp$).

At a high level, as a first step we will design a $\text{YOSO}^{\mathsf{WCC}}$-friendly VSS scheme. Then, same as in the PVSS-based construction, we will let $t+1$ dealers each share their secret randomness using this VSS. However, as mentioned above, this time we cannot rely on the public verifiability property of the secret sharing scheme. We used this property in the previous construction to determine whether a dealer behaved honestly during the commit/secret sharing phase. This, in turn, allowed us to circumvent the issue where a malicious party commits to some randomness, and *after seeing honest values* decides whether to open this randomness or not. In VSS, the dealer is allowed to send shares to the parties *privately*, and thus when a dealer and a share recipient are in dispute, from the perspective of an external party, it may not be immediately possible to tell whether the dealer or the share recipients behaves maliciously. Handling this requires further interaction and results in more roles in our $\text{YOSO}^{\mathsf{WCC}}$ VSS-based protocol, which we outline in the following.

We build our protocol around the well-known Pedersen VSS [14]. Recall that the standard stateful version of this VSS proceeds in the following four rounds, where s is the secret that is being shared, and g, h are generators of a group where computing a discrete logarithm is hard:

1. The dealer D chooses two degree-t polynomial

$$f_1(x) = a_0 + a_1 x + \cdots + a_t x^t,$$

$$f_2(x) = b_0 + b_1 x + \cdots + b_t x^t,$$

   such that $b_0 = s$. Then, D broadcasts commitments

$$(c_0, c_1, \cdots, c_t) = (g^{a_0} h^{b_0}, g^{a_1} h^{b_1} \cdots, g^{a_t} h^{b_t}),$$

   and sends $r_i = f_1(i)$, $s_i = f_2(i)$ to each P_i, $i \in [n]$.
2. Each party R_i checks whether $g^{r_i} h^{s_i} = \prod_{k=0}^{t} c_k^{i^k}$. If not, R_i broadcasts $\mathsf{Complain}$.
3. D broadcasts all shares from parties who complained. If any share that D broadcasts does not satisfy the above relation, D is deemed corrupt and the execution halts. Otherwise, each P_i who complained replaces its old share with the new (r_i, s_i).
4. Each R_i outputs r_i, s_i. The value $s = f_2(0)$ is the reconstructed secret.

Note that in the construction above, the dealer as well as each share recipient R_i may need to speak twice: The dealer is required to come back in the third round to resolve the complains, and each R_i might complain in the second round, and is then required to output its share in the fourth round. We adapt this scheme to the $\text{YOSO}^{\mathsf{WCC}}$ setting in two steps: First, we use two roles D, D' for the dealer, and let D not only execute the first round of the protocol above, but also send it state privately to D'. Second, we use two roles R_i, R_i' for each share

recipient, and also let R_i not only execute the round two of the protocol above, but send its state to its counterpart R'_i.

A final issue remains: Currently, we assume that g, h are publicly known values, and the construction above is secure assuming that $\log_g h$ is *not known to any party*. We would now like to remove this setup. The strawman idea is to simply have each dealer supply its own pair of g and h. However, if a malicious dealer colludes with a party R_i, then R_i can cheat by providing an invalid opening (which still verifies correctly), and thus changing the reconstructed secret value. To fix this, we substitute computationally Pedersen commitments by *unconditionally binding* ElGamal commitments. In more detail, we now compute the commitment $(c_0, c_1, \cdots, c_t)$ as follows:

$$(c_0, c_1, \cdots, c_t) = \big((g^{a_0}, h^{a_0} \cdot g^{b_0}), (g^{a_1}, h^{a_1} \cdot g^{b_1}), \cdots, (g^{a_t}, h^{a_t} \cdot g^{b_t})\big).$$

Pipelining. When implemented naively, the construction outlined above requires $6t + 4$ roles, and the construction is secure in the sending leaks model. To further decrease the number of roles, we carefully parallelize the execution of both dealer roles with the receiver roles. More concretely, instead of letting the $t + 1$ dealers to share secrets towards a fixed set of $2t + 1$ receivers, the recipient set for the i-th dealer is set to be the $2t + 1$ roles that immediately succeed that particular dealer. Moreover, observe that the conflicts regarding the i-th dealer can also be immediately resolved after the corresponding set of $2t + 1$ receiver roles have been executed.

This means that the total number of roles (after resolving complaints from all dealers) is now $3t + 3$, i.e. this linearization of roles allows us to decrease the total number of roles by roughly t in the sharing phase of the dealers. See further the details in Section 3.

Further optimization for execution leaks. We make the further observation that in the execution leaks model, we can further reduce the final set of receivers R'_i by t roles. The idea is that each original receiver R_i (from the sharing phase) follows the procedure of round two, but in addition sends its shares to *all* roles R'_j (instead of only R'_i as before), if its shares verify correctly. This step does not reveal information on the shares, since the channels to the future roles do not reveal any information until the corresponding recipient role is executed. In the reconstruction step, we can let each R'_j publish the received shares from all parties that it got the shares from. See Appendix C for details.

We arrive to the final theorem.

Theorem 2 (informal). *Assuming ElGamal commitments, there exists a computationally secure randomness generation protocol with* $5t+4$ *roles (resp.* $4t+4$*) in the sending-leaks (resp. execution-leaks model, where* t *is the number of corruptions.*

2 PVSS-based YOSO^{WCC} Randomness Generation

We introduce a randomness generation scheme which relies on publicly verifiable secret sharing (PVSS). Before going into our protocol, we briefly explain what a PVSS is.

2.1 Publicly Verifiable Secret Sharing

Recall the definition of Publicly Verifiable Secret Sharing (PVSS) from [4]. In PVSS, a dealer D shares a secret to a set of n parties $\mathcal{P} = \{P_1, \cdots, P_n\}$. A (t, n)-PVSS protocol ensures that a secret is split in a way that allows $t + 1$ parties to reconstruct a secret, but at the same time, knowing t shares does not reveal any information about the secret. Any external verifier V is able to check that D acts honestly. More formally, a PVSS protocol consists of the algorithms $(\mathsf{Setup}, \mathsf{Dist}, \mathsf{Verif}, \mathsf{Reconstr\text{-}Dec}, \mathsf{Reconstr\text{-}Pool})$, where $\mathsf{Setup} = (\mathsf{Setup}_\pi, \mathsf{Setup}_{\mathsf{PKI}})$, and which denote the following:

- **Setup:** Consists of $(\mathsf{Setup}_\pi, \mathsf{Setup}_{\mathsf{PKI}})$, which take security parameter λ as input. In Setup_π, the parameters of the proof system are generated in a trusted fashion. Using $\mathsf{Setup}_{\mathsf{PKI}}$, every party generates a public key pk_i and withholds the corresponding secret key sk_i.
- **Distribution:** The dealer creates shares $s_1, \cdots, s_n$ for the secret s, encrypts share s_i with the key pk_i for $i = \{1, \cdots, n\}$ and publishes these encryptions $\hat{s}_i$, together with a proof $\textsc{proof}_D$ that these are indeed encryptions of a valid sharing of some secret.
- **Verification:** In this phase, any external V (not necessarily being a participant in the protocol) can verify non-interactively, given all the public information until this point, that the values $\hat{s}_i$ are encryptions of a valid sharing of some secret.
- **Reconstruction:** This phase is divided in two.
  Decryption of the shares: This phase can be carried out by any set Q of $t+1$ or more parties. Every party P_i in Q decrypts the share s_i from the ciphertext $\hat{s}_i$ by using its secret key sk_i, and publishes s_i together with a (non-interactive) zero-knowledge proof $\textsc{proof}_i$ that this value is indeed a correct decryption of $\hat{s}_i$.
  Share pooling: Any external verifier V (not necessarily being a participant in the protocol) can now execute this phase. V first checks whether the proofs $\textsc{proof}_i$ are correct. If the check passes for less than $t + 1$ parties in Q then V aborts; otherwise V applies a reconstruction procedure to the set s_i of shares corresponding to parties P_i that passed the checks.

A PVSS protocol $(\mathsf{Setup}, \mathsf{Dist}, \mathsf{Verif}, \mathsf{Reconstr\text{-}Dec}, \mathsf{Reconstr\text{-}Pool})$ must provide three security guarantees: Correctness, Verifiability and IND1-Secrecy. These properties are defined below:

- **Correctness:** If the dealer and all players in Q are honest, then all checks in the verification and reconstruction phases pass, and the secret can be reconstructed from the information published by the players in Q during reconstruction.
- **Verifiability:** If the check in the Verification phase passes, then with high probability the values $\hat{s}_i$ are encryptions of a valid sharing of some secret. Furthermore, if the check in the Reconstruction phase passes, then the values s_i are indeed the shares of the secret distributed by D.
- **IND1-Secrecy:** Prior to the reconstruction phase, the public information together with the secret keys sk_i of any set of at most t players gives no information about the secret.

2.2 Our PVSS-Based Randomness Generation Protocol

Our protocol is in the sending-leaks model (thus also secure for execution-leaks). We describe the scheme and outline the security proof, for full proof see Appendix B.

The high-level idea of the scheme is the following: given $n = 3t + 2$ parties, split them into two groups $\mathcal{P}$ and $\mathcal{P}'$ of size $t + 1$ and $2t + 1$, respectively. We dub the parties from the first group *dealers*, denoted by $P_1, P_2, \cdots, P_{t+1}$, and the parties from the second group *decryptors*, denoted by $P'_1, P'_2, \cdots, P'_{2t+1}$. Let $(\mathsf{Setup} := (\mathsf{Setup}_\pi, \mathsf{Setup}_{\mathsf{PKI}}), \mathsf{Dist}, \mathsf{Verif}, \mathsf{RDec}, \mathsf{RPool})$ denote a $(t, 2t + 1)$-PVSS protocol. The protocol starts with a "sharing" phase, where every P_i is executed one after another and acts as a PVSS dealer distributing its secret to the decryptors in $\mathcal{P}'$. Then, decryptors $P'_i \in \mathcal{P}'$ are executed one after another, and each decryptor P'_i executes the share decryption part of the PVSS reconstruction phase for each dealer P_i. Finally, any party C can execute the share pooling phase of the PVSS reconstruction phase in order to obtain the secret shared by each dealer. We give the full scheme in Protocol 1.

For security, we need our PVSS to be non-malleable, which can be naively achieved by using simulation-extractable NIZKs [9] as PVSS proofs. Intuitively, a strawman PVSS scheme which provides the required non-malleability works as follows: Share the secret using a (t, n) secret sharing scheme (e.g, Shamir's secret sharing [16]), encrypt each share using a public key of the corresponding share receiver, and append a simulation-extractable NIZK proof confirming that the dealer knows the shares underlying the ciphertexts, and these shares correspond to the (t, n) secret sharing. The reconstruction works by having each receiver decrypt its share, and publish a proof confirming that it knows a secret key such that the decryption of the corresponding ciphertext results in the stated value. See Appendix A for details. The communication complexity is $O(n^2|c| + n|p|)$, where $|c|$ is the length of a single ciphertext, and $|p|$ of a proof.

Theorem 3. *Assuming public key encryption and simulation-extractable NIZKs, there exists a YOSO^{WCC} $(t, 3t + 2)$-computationally secure randomness generation protocol in the sending-leaks model.*

Protocol 1 Randomness Beacon from PVSS in the Sending-Leaks Model.

Setup: PVSS Setup_π algorithm is executed in a trusted fashion to obtain the common reference string crs. Public key of every party in the protocol is generated according to $\mathsf{Setup}_{\mathsf{PKI}}$.

Sharing phase: Each party P_i, $i \in [t+1]$ does the following:

1. P_i samples x_i from $\{0,1\}$ uniformly at random.
2. P_i uses PVSS algorithm Dist as the dealer to distribute shares of x_i to the parties $P'_1, \cdots, P'_{2t+1}$:

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}) \leftarrow \mathsf{Dist}(x_i, \{pk_{P'_j}\}_{j\in[2t+1]}, \mathsf{crs}).$$

3. P_i publishes $(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)})$.

Reconstruction phase: Each party P'_j, $j \in [2t+1]$ does the following:

1. For each P_i, P'_j uses $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs})$ to verify that P_i dealt a valid secret. For each P_i who passed the check, P'_j verifies that the proof $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.
2. For each valid P_i, P'_j uses the PVSS algorithm $\mathsf{RDec}(\hat{s}_j^{(i)}, sk'_{P'_j}, \mathsf{crs})$ to obtain $(s_j^{(i)}, \textsc{proof}_j^{(i)})$, and publishes this pair.

Any party C can use the PVSS algorithm RPool on information published by the parties $P'_1, \cdots, P'_{2t+1}$ to obtain x_i. Output $\bigoplus_{i\in I} x_i$, where I denotes an index set of dealers for which C obtained the secret using RPool.

Intuitively, the security of the scheme follows from the security of the PVSS scheme and the proof system used. The security argument uses the standard hybrid argument and formal proof can be found in Appendix B. By the zero-knowledge property of the NIZK we can change the proofs of the honest parties to the simulated ones. Then, we use the extractor of the NIZK to extract the witnesses of the proofs (which verify correctly) submitted by the adversarial dealers, aborting whenever extraction fails. By the simulation extractability of the NIZK, the abort happens only with a negligible probability. We then use the extracted values instead of the ones that were obtained during the reconstruction phase as the contribution of the adversarial parties in the computation of the coin output of the protocol. By the verifiability property of PVSS, the extracted values are exactly the ones obtained during the reconstruction, thus the difference between this and the previous hybrid is negligible. Next, for honest dealers, instead of using the values obtained during the reconstruction, we use the dealers' initial secrets distributed in the computation of the coin output of the protocol. By the correctness of PVSS, the difference between this and the last hybrid is negligible. Now, note that the coin output of the protocol which is given to the distinguisher is fixed and known by the end of the sharing phase. We can now use the IND1-secrecy property of PVSS to change the ciphertexts distributed by the honest dealers during the sharing phase to encryptions of

the sharings of newly generated uniformly random values (while continuing to use the honest values x_i in the computation of the coin output). Note that in the final hybrid the coin output of the protocol does not depend on the secrets generated by the honest parties.

3 Randomness Generation from ElGamal Commitments

We now describe our randomness generation protocol that is secure against computational adversaries, and *does not require any setup assumptions*. We provide two variations of this protocol: One for the sending-leaks, the other for the execution-leaks model. To reduce the number of roles, we use pipelining in both versions. For simplicity, we first describe the protocol without the pipelining.

Sending leaks construction As mentioned in Section 1.3, the high-level idea of the construction is the following: as a first step, we "linearize" a custom version of the Pedersen's [14] VSS protocol where a single party shares a random value in our stateless model. Recall that in each linearization we have the roles:

- Party D, who acts as the dealer distributing the secrets (publishing commitments to the coefficients of t-degree polynomial and bilaterally sending to each receiver a share evaluation), and sends its state to its counterpart D'.
- $2t+1$ receivers R_i, who receive and verify the secret shares, complain about the shares if applicable, and otherwise send these to the counterpart R'_i.
- Party D' who obtains a state from D and uses it to publish the shares of the receivers that complained.
- $2t+1$ receivers R'_i who receive the shares from their counterparts R_i, as well as set their shares to the ones broadcast by D' (if the counterpart R_i complained), and publicly reveal these shares.

We use $t+1$ such linearized VSS to share $t+1$ random values, and output the final coin as the xor the results. In more detail, we let $n = 6t+4$, and divide the n parties into a group $\mathcal{D}$ of size $t+1$, group $\mathcal{R}$ of size $2t+1$, group $\mathcal{D}'$ of size $t+1$, and group $\mathcal{R}'$ of size $2t+1$. These parties execute the following roles:

- Each $D_i \in \mathcal{D}$ acts as the dealer D in the i-th linearization.
- Each $R_i \in \mathcal{R}$ executes the role the i-th receiver R_i in *each* of the $t+1$ linearizations.
- Each $D'_i \in \mathcal{D}'$ acts as the dealer D' in the i-th linearization.
- Each $R'_i \in \mathcal{R}'$ executes the role of the i-th receiver R'_i in *each* of the $t+1$ linearizations.

We denote a client who wishes to obtain the result of the protocol by C (C can be external, but can also be one of D_i, R_i, D'_i, R'_i).

The protocol starts with a "sharing" phase, where each D_i is executed one after another and shares its secret via Shamir's secret sharing to the receivers in $\mathcal{R}$, while committing to it using ElGamal's commitments. Additionally, each

D_i sends its state to its counterpart D'_i. Then, parties in $\mathcal{R}$ are executed one after another, verify the shares they receive, and complain about the dealers who sent inconsistent shares. Finally, each dealer D'_i uses the state it received from its counterpart D_i to publicly respond to the complains. After this, the sharing phase is completed and the "reconstruction phase" begins. Here, each $R'_i \in \mathcal{R}'$ simply outputs the shares it received. Every party C who is interested in the output verifies the published shares, uses the ones that passed the verification to reconstruct the secret dealt by a particular dealer, and computes the xor of all secrets dealt by the dealers who were not deemed corrupt (i.e., publicly sent inconsistent information as a respond to a complain during the sharing phase). See Protocol 2 for details.

Execution leaks variation For the execution leaks model, we similarly implement the behavior of each dealer using two roles – one responsible for the sharing of a secret, and one responsible for addressing the complaints. However, instead of implementing each R_i using two roles, we have $2t+1$ parties R_i and $t+1$ parties R'_j (where R'_i can *not* be thought of as a counterpart of R_i). Each R_i follows the procedure of round two, and if its shares verify, it additionally sends its shares to *each* R'_j. Finally, each R'_j publishes all shares (from all parties got the shares from) which verified correctly. See Appendix C for details.

Pipelining optimization Implemented naively, in the sending leaks model the protocol described above requires $6t+4$ parties, and its execution leaks variation requires $5t+4$ parties. To reduce this number, we propose the following modification to both the sending leaks, and the execution leaks protocol: Instead of combining multiple linearized VSS by having $t+1$ dealers, each of whom shares secrets among the *same set* $\mathcal{R}$ of $2t+1$ parties, we let each dealer share secrets among the *next* $2t+1$ parties. In the sending leaks model, we now have $n = 5t+4$ parties P_i, where depending on the index i, party P_i executes the following roles:

- For $1 \leq i \leq t+1$, party P_i executes the role of the dealer D in i-th VSS linarization. If additionally $i > 1$, P_i also executes the role R_{i-j} in j-th VSS linearization, where $j < i$.
- For $t+2 \leq i \leq 3t+2$, party P_i executes the role R_{i-j} in j-th VSS linearization, where $j < i$. If additionally $i > 2t+2$, P_i also executes the role of the dealer D' in the $i-2t-2$-th VSS linearization.
- For $i = 3t+3$, party P_i executes the role D' in the $t+1$-st VSS linearization.
- For $3t+4 \leq i \leq 5t+4$, P_i executes the role R'_{i-3t-3} for each linearization.

Theorem 4. *Assuming ElGamal commitments, there is a YOSO*^{WCC} $(t, 5t+4)$*-secure computational randomness generation protocol in the sending-leaks model.*

Proof. Consider a PPT adversary A biasing the coin output of the protocol, and a PPT distinguisher D that is able to distinguish the protocol output from a uniformly random coin with probability $\frac{1}{2} + \delta$, where δ is non-negligible. Note

that A controls at most t parties among the share recipients. Denote these parties by P_i with $i \in [t]$. We construct an adversary B on the hiding property of the underlying commitment scheme as follows. First, B samples two inputs r_0, r_1 at random. Next, B sends these to its challenger C, and obtains a commitment (g^*, h^*). B samples t pairs of r_i and s_i, and computes commitments $(c_0^*, c_1^*, \cdots, c_t^*)$ consistently with (g^*, h^*) as well as the t pairs r_i, s_i via Lagrange interpolation in the exponent. Now, note that in our protocol there are $t+1$ dealers, thus at least one dealer must be honest. B acts honestly in its interaction with A, except that for one honest dealer B behaves as follows: B publishes the previously computed commitments $(c_0^*, c_1^*, \cdots, c_t^*)$. Then, B sends the previously sampled pairs s_i, r_i to the corresponding adversarial parties. The honest P_i's controlled by B do not complain about not receiving their shares. If A complains about a share, B publishes the corresponding share s_i, r_i according to the protocol. Now, note that as B controls $t+1$ parties P_i, the information submitted by A prior to the start of the reconstruction phase is sufficient to compute the coins dealt by A (or discard a subset of such coins if some dealers are deemed corrupt). Let r_A denote the XOR of all valid coins dealt by A. Now, B sends $r_0 \oplus r_A$ to D. If D's verdict is $\mathsf{Protocol\ output}$, B sends 0 to its challenger C. Otherwise, B sends 1 to C. Let b^* denote the index chosen by the challenger C. B is correct with probability $\frac{1}{2} + \delta$, for non-negligible δ, and therefore B breaks computational hiding of the commitment with non-negligible probability, which is a contradiction.

By an analogous proof, we obtain the following result for the execution-leaks model:

Theorem 5. *Assuming ElGamal commitments, there is a YOSO*^{WCC} $(t, 4t+4)$*-secure computational randomness generation protocol in the execution-leaks model.*

4 Implementation and Evaluation

We now evaluate our randomness generation scheme in the sending leaks model from Section 3. In the following, we first compare it to our implementation of the randomness extraction protocol from [13]. We then evaluate the work required to be performed by each role in Appendix D. Our implementation is available at https://github.com/yosorand/yoso-rand-elgamal and required ∼300 LoC in Rust. We ran all our experiments single-threaded on a MacBook Pro with 32GB of RAM, and an Apple M1 Pro SoC.

In our proof-of-concept implementation, we simulate the communication layer (i.e., the broadcast and point-to-point channels), and assume that the channels are authenticated. In our implementation all parties behave honestly, which corresponds to the worst case in terms of communication and computation complexity (same for NRO).

In terms of the running times (see Table 1), as expected, for very small values of t the NRO protocol is faster than us. However, due to the exponential

Protocol 2 SL Randomness Generation from ElGamal Commitments

Sharing phase:
Each D_i, $i \in [t+1]$ does the following:

1. D_i chooses random degree-t polynomials f_1 and f_2:
$$f_1 = a_0 + a_1 x + \cdots + a_t x^t \text{ and } f_2 = b_0 + b_1 x + \cdots + b_t x^t.$$
2. D_i chooses a pair of generators (g, h).
3. D_i commits to f_1 and f_2 via broadcasting (g, h) along with
$$(c_0, c_1, \cdots, c_t) = \Big((g^{a_0}, h^{a_0} \cdot g^{b_0}), (g^{a_1}, h^{a_1} \cdot g^{b_1}), \cdots, (g^{a_t}, h^{a_t} \cdot g^{b_t})\Big).$$
4. D_i sends $r_j = f_1(j)$ and $s_j = f_2(j)$ to each P_j, $j \in [2t+1]$; and sends polynomials f_1 and f_2 to D'_i.

Each R_i, $i \in [2t+1]$ does the following:

1. For each dealer D_j, R_i checks whether the share (r_i, s_i) it obtained from D_j, and the commitments to f_1 and f_2 distributed by D_j satisfy
$$g^{r_i} = \prod_{k=0}^{t} (g^{a_k})^{i^k} \text{ and } h^{r_i} \cdot g^{s_i} = \prod_{k=0}^{t} \Big(h^{a_k} \cdot g^{b_k}\Big)^{i^k}$$
If not, R_i broadcasts $\mathsf{Complain} - \mathsf{D_j}$.
2. R_i sends all shares (r_i, s_i) that passed verification to R'_i.

Each D'_i, $i \in [t+1]$ does the following:

1. D'_i broadcasts shares of parties who complained about D_i. If any share broadcast by D'_i does not pass the check above, D'_i is deemed corrupt.

Reconstruction phase:
Each R'_i, $i \in [2t+1]$ does the following:

1. If R_i complained about D_j, and D'_j was not deemed corrupt, R'_i sets its corresponding share to s_i and r_i broadcast by D'_j.
2. R'_i outputs all shares (s_i, r_i) it obtained for non-corrupt dealers.

Client C does the following:

1. For each D'_i who was not deemed corrupt, C uses any $t+1$ shares s_j and r_j that pass the verification check against the corresponding commitment to reconstruct the value $s_i = f_i(0)$, where f_i is the polynomial f_2 dealt by D_i/D'_i.
2. Let H denote the index set of dealers D'_i which were not deemed corrupt. C outputs $\bigoplus_{i \in H} s_i$.

computational complexity of the NRO protocol, we outperform NRO already for $t = 6$, and the NRO scheme becomes impractical for values as small as $t = 8$. This gap will only increase as t grows. Further, note that for us, Client's work

(which is off the critical path) takes approximately one third of the overall time (see Appendix D, Client's overall work in the randomness generation requires time which is roughly $(t+1)$ times the Client's execution time for a single VSS).

t	Our scheme	NRO
1	314	0.64
2	821	2
3	1589	24
4	2793	236
5	4285	2463
6	6312	24387
7	8886	233328
8	11966	-

Table 1: Running time comparison, all times in milliseconds.

t	Our scheme	NRO
1	0.0031	0.0003
2	0.0067	0.004
3	0.0115	0.039
4	0.0176	0.378
5	0.0249	3.399
6	0.0336	29.182
7	0.0436	242.327
8	0.0548	-

Table 2: Overall communication sizes in MB.

Note that while in our evaluation we assume that all parties are single-threaded, our scheme is easily parallelizable: As a party often executes multiple roles for *independent* protocol executions of the linearized VSS, those roles can be executed by different cores. This slashes the cost roughly by a factor which corresponds to the number of cores available.

Finally, we report the overall data sizes that parties need to transmit both in our protocol and the NRO protocol (see Table 2). Again, while the NRO protocol is very efficient on very small values of t, our scheme outperforms it already for $t = 3$ (and stays remarkably low for larger values of t). This gap will only grow, as the NRO protocol has exponential communication complexity. As before in our running time experiment, we did not obtain the final values for the NRO scheme due to the timeout.

Acknowledgements

This work was supported by a Protocol Labs Cryptonet Network Grant "Stateless Distributed Randomness Generation". J. Ribeiro's research was also supported by NOVA LINCS (ref. UIDB/04516/2020) with the financial support of FCT - Fundação para a Ciência e a Tecnologia. Elisaweta Masserova was supported by a gift from Bosch and NSF Grants No. 1801369 and 2224279. We thank Jay Bosamiya for helping us with the implementation of this work.

References

1. Baignères, T., Delerablée, C., Finiasz, M., Goubin, L., Lepoint, T., Rivain, M.: Trap me if you can – million dollar curve. IACR Cryptol. ePrint Arch. (2015), http://eprint.iacr.org/2015/1249

2. 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. pp. 260–290. Springer International Publishing, Cham (2020)
3. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Advances in Cryptology – CRYPTO 2018. pp. 757–788. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_25
4. Cascudo, I., David, B.: SCRAPE: Scalable Randomness Attested by Public Entities. In: International Conference on Applied Cryptography and Network Security. pp. 537–556 (2017)
5. Cascudo, I., David, B.: ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In: Advances in Cryptology – ASIACRYPT 2020. pp. 311–341. Springer International Publishing, Cham (2020)
6. Choi, K., Manoj, A., Bonneau, J.: SoK: Distributed randomness beacons. In: 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023. pp. 75–92. IEEE (2023). https://doi.org/10.1109/SP46215.2023.10179419
7. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You Only Speak Once. In: Annual International Cryptology Conference. pp. 64–93 (2021)
8. Gentry, C., Halevi, S., Magri, B., Nielsen, J.B., Yakoubov, S.: Random-index PIR and applications. In: Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13044, pp. 32–61. Springer (2021)
9. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) Advances in Cryptology – ASIACRYPT 2006. pp. 444–459. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
10. Groth, J.: Non-interactive distributed key generation and key resharing. IACR Cryptol. ePrint Arch. (2021), https://eprint.iacr.org/2021/339
11. Kelsey, J., Brandão, L.T.A.N., Peralta, R., Booth, H.: A reference for randomness beacons: Format and protocol version 2. Tech. rep., National Institute of Standards and Technology (2019), https://csrc.nist.gov/pubs/ir/8213/ipd
12. Lenstra, A.K., Wesolowski, B.: A random zoo: sloth, unicorn, and trx. IACR Cryptol. ePrint Arch. (2015), http://eprint.iacr.org/2015/366
13. Nielsen, J.B., Ribeiro, J., Obremski, M.: Public randomness extraction with ephemeral roles and worst-case corruptions. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 127–147. Springer Nature Switzerland, Cham (2022)
14. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91. pp. 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg (1992)
15. Rabin, M.O.: Transaction protection by beacons. Journal of Computer and System Sciences **27**(2), 256–267 (1983)
16. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
17. Stadler, M.: Publicly verifiable secret sharing. In: Maurer, U.M. (ed.) Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding. Lecture Notes in Computer Science, vol. 1070, pp. 190–199. Springer (1996)

18. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 444–460 (2017). https://doi.org/10.1109/SP.2017.45
19. Thyagarajan, S.A.K., Castagnos, G., Laguillaumie, F., Malavolta, G.: Efficient CCA timed commitments in class groups. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 2663–2684. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484773

Appendix

A A Note on PVSS and NIZKs

We now further outline the security properties of PVSS we rely on in our PVSS-based randomness generation scheme. Recall that a PVSS scheme consists of a tuple of algorithms

$$(\mathsf{Setup}, \mathsf{Dist}, \mathsf{Verif}, \mathsf{Reconstr\text{-}Dec}, \mathsf{Reconstr\text{-}Pool}),$$

where $\mathsf{Setup} = (\mathsf{Setup}_\pi, \mathsf{Setup}_{\mathsf{PKI}})$. We first formally specify the IND-1 secrecy of the scheme:

Definition 2. *We say that the PVSS is* IND1-secret *if for any PPT adversary A corrupting at most t parties, A has negligible advantage in the following game played against a challenger C.*

1. *C runs the Setup phase of the PVSS and sends all public information to A. Moreover, it creates secret and public keys for all uncorrupted parties, and sends the corresponding public keys to A.*
2. *A creates secret keys for the corrupted parties and sends the corresponding public keys to C.*
3. *C chooses values x_0 and x_1 at random in the space of secrets. Furthermore it chooses $b \leftarrow \{0,1\}$ uniformly at random. It runs the Distribution phase with x_0 as secret. It sends A all public information generated in that phase, together with x_b.*
4. *A outputs a guess $b' \in \{0,1\}$.*

The advantage of A is defined as $|\Pr[b = b'] - \frac{1}{2}|$.

In addition to the above IND-1 secrecy, as well as correctness and verifiability, which have been defined previously, we require our PVSS to be *non-malleable*.

A Note on Non-Malleability. To obtain the non-malleability guarantee required by our construction we informally require the compatibility with the (unbounded) computational zero-knowledge property and the simulation-extractability property of the underlying proof system. In more detail, consider the proof system (G, P, V) for a relation R, where

- $\sigma \leftarrow G(1^\lambda)$: given a security parameter λ, the key generation algorithm produces a crs σ.
- $\pi \leftarrow V(\sigma, x, w)$: the prover algorithm takes as input a crs σ, a statement x, and a witness, and produces a proof π.
- $b \leftarrow G(\sigma, x, \pi)$: the verifier algorithm takes as input a crs σ, a statement x produces a crs σ.

We consider non-interactive proofs, and require that in addition to the standard completeness and soundness guarantees, the proof system has the following properties:

Definition 3 ((unbounded) computational zero-knowledge). *A non-interactive proof system (G, P, V) is zero-knowledge for a relation R, if there exists a PPT simulator consisting of a tuple of PPT algorithms $S = (S_1, S_2)$, such that for all PPT adversaries A holds that*

$$Pr|[\sigma \leftarrow G(1^\lambda) : A^{P(\sigma,\cdot,\cdot)}(\sigma) = 1] \\ - Pr[(\sigma, \tau) \leftarrow S_1(1^\lambda) : A^{S(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1]| < negl(\lambda)$$

where $S(\sigma, \tau, x, w) = S_2(\sigma, \tau, x)$ if $(x, w) \in R$.

We call non-interactive zero-knowledge proof systems *NIZKs*. We additionally require the following:

Definition 4 (Simulation Extractability [9]). *We call a NIZK (G, P, V) simulation-extractable if there exists a tuple of PPT algorithms (SE_1, E), such that SE_1 output a triple (σ, τ, ζ), which is identical to the output of S_1 when restricted to the first two parts, and for all PPT adversaries A holds that*

$$\Pr\left[\begin{array}{cc} (\sigma,\tau,\zeta) \leftarrow SE_1(1^\lambda) & (x,\pi) \notin Q \\ (x,\pi) \leftarrow A^{S_2(\sigma,\tau,\cdot)}(\sigma,\zeta) : & (x,w) \notin R \\ w \leftarrow E(\sigma,\zeta,x,\pi) & V(\sigma,x,\pi) = 1 \end{array}\right] < negl(\lambda),$$

where Q is the list of A's simulation queries and responses.

Additionally, we require that the proof system is "decoupled" from the encryption scheme used in the PVSS, in the sense that the keys and the proof crs are generated independently of each other, and the distribution algorithm can be split into two steps, first of which produces the ciphertexts $\hat{s}_i$, and the second of which produces a NIZK proof given these ciphertexts.

Note that such PVSS scheme can be trivially built from a public-key encryption scheme and a simulation-extractable NIZK as follows. First, the dealer splits its secret using a (t, n) Shamir secret sharing, and encrypts each share using the public keys of the share receivers. Then, the dealer generates a NIZK proof confirming that it knows shares underlying the ciphertexts, and these lie on a polynomial of degree at most t. Anyone can verify the correctness of the dealer's sharing using the verifier for the NIZK proof. In the reconstruction phase, every share recipient decrypts its share, and generates a proof that the decrypted value indeed corresponds to the ciphertext published by the dealer. Given $t+1$ honest share recipients, the correctness of the scheme follows from the correctness of the encryption scheme, completeness and soundness of the NIZK, and correctness of Shamir's secret sharing. Privacy follows from the security of the encryption scheme, zero-knowledge of the NIZK, and security of Shamir's secret sharing. Verifiability follows from the simulation-extractability of the NIZK, correctness of the encryption, and and the fact that any $t + 1$ shares fix the secret.

B PVSS-based YOSO^{WCC} Randomness Generation: Security Proof

We prove the theorem via a hybrid argument. In the following, let λ denote the security parameter.

Hybrid $\mathbf{H_0}$: This hybrid corresponds to the real world experiment as defined in Definition 1 with the bit b fixed to 0. Specifically, the challenger interacts with a PPT adversary A that corrupts a set M of parties, where $|M| = t$, and interacts with a set H, $|H| = 2t + 2$, of honest parties to obtain the coin output of the Protocol 1, and forwards this output to a PPT distinguisher D.

1. $\mathsf{crs} \leftarrow \mathsf{Setup}_\pi(1^\lambda)$.
2. For $P'_i \in H$, let $(pk'_i, sk'_i) \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P'_i \in M$, let $pk'_i \leftarrow A(\{pk'_j\}_{P'_j \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0, 1\}$ uniformly at random.
   (b) Publish
$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}) \leftarrow \mathsf{Dist}(x_i, \{pk'_j\}_{j\in[2t+1]}, \mathsf{crs}).$$
5. For $P'_i \in H$:
   (a) For each P_i such that
$$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs}) = 1,$$
   check whether $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.
   (b) For each valid P_i publish
$$(s_j^{(i)}, \textsc{proof}_j^{(i)}) \leftarrow \mathsf{RDec}(\hat{s}_j^{(i)}, sk'_i, \mathsf{crs}).$$
6. For each $i \in [t + 1]$ let
$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \textsc{proof}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$
7. $\mathsf{out} \leftarrow \bigoplus_{i\in I} \mathsf{out}_i$, where I denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$.
8. $b' \leftarrow D(\mathsf{out})$.

Here, M denotes the set of parties controlled by A, and H is the set of honest parties.

Hybrid $\mathbf{H_1}$: This hybrid is the same as before, except that instead of computing proofs $\textsc{proof}_D$ and $\textsc{proof}_i$ honestly, the proofs are generated using a simulator.

The game becomes the following (changes from the previous hybrid in red):

1. $(\mathsf{crs}, \tau) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j' \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P_i' \in H$:
   (a) For each P_i such that $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs}) = 1$, check whether $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.
   (b) For each valid P_i publish

$$(s_j^{(i)}, \textsc{proof}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}, \tau).$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \textsc{proof}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in I} \mathsf{out}_i$, where I denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$.
8. $b' \leftarrow D(\mathsf{out})$.

Lemma 1. *Assuming that the proof system used in the PVSS scheme has the zero-knowledge property, the outputs of experiments* $\mathbf{H_0}$ *and* $\mathbf{H_1}$ *are computationally indistinguishable.*

Proof. This is a series of hybrids in which each honest proof is replaced one by one. Given a PPT adversary A and a distinguisher D who is able to distinguish between the two hybrids given the output of the challenger's interaction with A, we construct an adversary B on the zero-knowledge property of the underlying PVSS scheme as follows. B obtains the setup information for the proof system used in PVSS from its challenger C, and forwards this information to the adversary A. Then, B follows the game as specified by the previous hybrid (using the challenger to obtain simulated proofs if required by the previous hybrid), except that when B must produce the proof $\textsc{proof}_D$ ($\textsc{proof}_i$), B uses the simulated proof which it obtains from C. B forwards the protocol output of its interaction with A to D. If D outputs "Hybrid 0", B outputs "Real prover", otherwise "Simulator". As the only difference between the two hybrids is the way that $\textsc{proof}_D$ ($\textsc{proof}_i$) is being generated, B's advantage is the same as D's. Thus, if the advantage of the pair A and D is non-negligible, B's advantage is non-negligible as well.

Hybrid $\mathbf{H_2}$: This hybrid is the same as before, except that the challenger uses the extractor Ext to extract the corresponding secret from each $\textsc{proof}_D$ that verifies correctly. The challenger aborts if the extraction fails.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimExtSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j' \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish
$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$
5. For $P_i' \in H$:
   (a) For each P_i such that
$$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs}) = 1,$$
   check whether $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.
   (b) For each valid $P_i \in M$ let $w \leftarrow \mathsf{Ext}(\textsc{proof}_D^{(i)}, \mathsf{crs}, \zeta)$. If
$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, w) \notin R,$$
   then abort.
   (c) For each valid P_i publish
$$(s_j^{(i)}, \textsc{proof}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}, \tau).$$
6. For each $i \in [t+1]$ let
$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk_i'\}_{i\in[2t+1]}, \textsc{proof}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$
7. $\mathsf{out} \leftarrow \bigoplus_{i\in I} \mathsf{out}_i$, where I denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$.
8. $b' \leftarrow D(\mathsf{out})$.

Lemma 2. *Assuming that the proof system used in the PVSS scheme has the simulation extractability property, the outputs of experiments* $\mathbf{H_1}$ *and* $\mathbf{H_2}$ *are computationally indistinguishable.*

Proof. This is a series of hybrids in which each adversarial proof which verifies correctly is handled one by one. In the i-th such hybrid step, given a PPT adversary A and a distinguisher D who is able to distinguish between the two hybrids given the output of the challenger's interaction with A, we construct an adversary B on the simulation extractability of the underlying PVSS scheme as follows. B obtains the setup information for the proof system used in PVSS

from its challenger C, and forwards this information to the adversary A. Then, B follows the game as specified by the previous hybrid, except that it uses C to obtain simulated proofs that B is required to generate according to the protocol. When the adversary A publishes the i-th adversarial proof, B forwards this proof to its challenger C, and uses the extractor on this proof. If the extraction succeeds, B forwards the protocol output of its interaction with A to D, otherwise B aborts. Note that the two hybrids are exactly the same when the proof extraction succeeds. Thus, we get that

$$\begin{aligned}
&\Pr[(A,D) \text{ wins}] \\
&= \Pr[(A,D) \text{ wins}|\text{Extr. succeeds}] \cdot \Pr[\text{Extr. succeeds}] \\
&+ Pr[(A,D) \text{ wins}|\text{Extr. fails}] \cdot \Pr[\text{Extr. fails}] \\
&= \left(\frac{1}{2} + \mathsf{negl}(\lambda)(\lambda)\right)(1 - \Pr[\text{Extr. fails}]) \\
&+ \Pr[(A,D) \text{ wins}|\text{Extr. fails}] \cdot \Pr[\text{Extr. fails}] \\
&\leq \frac{1}{2} + \mathsf{negl}(\lambda)(\lambda) + Pr[\text{Extr. fails}]\left(1 - \frac{1}{2} - \mathsf{negl}(\lambda)(\lambda)\right).
\end{aligned}$$

Therefore, we have that

$$\Pr[\text{Extr. fails}] \geq \frac{\Pr[A \text{ wins}] - \frac{1}{2} - \mathsf{negl}(\lambda)(\lambda)}{\frac{1}{2} - \mathsf{negl}(\lambda)(\lambda)}.$$

Note that B wins whenever the extraction fails. Thus, if A wins with some non-negligible advantage, B wins with non-negligible probability as well.

Hybrid $\mathbf{H_3}$: This hybrid is the same as before, except that the protocol outcome computation is modified as follows: For the dealers *controlled by the adversary* which pass the check of the PVSS verification phase, instead of using the secrets obtained for these dealers during the reconstruction phase, the challenger uses the secrets that were extracted using the extractor Ext.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j' \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P_i' \in H$:
   (a) For each P_i such that $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs}) = 1$, check whether $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.

(b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\text{PROOF}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If

$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, w) \notin R,$$

then abort.

(c) For each valid P_i publish

$$(s_j^{(i)}, \text{PROOF}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}, \tau)$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk_i'\}_{i\in[2t+1]}, \text{PROOF}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in I\cap H} \mathsf{out}_i \oplus \tilde{x}_{i\in\tilde{I}}$, where I denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$, $\tilde{I} \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds P_i' is valid, and $\tilde{x}_i$ is the secret corresponding to the witness w_i.
8. $b' \leftarrow D(\mathsf{out})$.

Lemma 3. *Assuming that the PVSS scheme is verifiable, the outputs of experiments* $\mathbf{H_2}$ *and* $\mathbf{H_3}$ *are computationally indistinguishable.*

Proof. This is a series of hybrids, where we change the contribution of each malicious dealer one-by-one. By the verifiability property of the PVSS scheme, if the verifications checks passes, then the sharing phase determines a unique secret, and this secret will be reconstructed by the end of the reconstruction phase. As the extractor Ext extracted a valid secret s^*, and the secret determined by the sharing phase is *unique* and is guaranteed to be reconstructed by the end of the protocol, s^* is exactly the secret that the parties would have reconstructed for this dealer by the end of the reconstruction phase.

Hybrid $\mathbf{H_4}$: This hybrid is the same as before, except that the protocol outcome computation is modified as follows: For the *honest* dealers, instead of using the secrets obtained for these dealers during the reconstruction phase, the challenger uses the secrets that these dealers shared during the sharing phase:

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j'\in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P_i' \in H$:

(a) For each P_i such that

$$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs}) = 1,$$

check whether $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.

(b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\textsc{proof}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If

$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, w) \notin R,$$

then abort.

(c) For each valid P_i publish

$$(s_j^{(i)}, \textsc{proof}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}, \tau).$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk_i'\}_{i\in[2t+1]}, \textsc{proof}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in H} x_i \oplus \tilde{x}_{i\in\tilde{I}}$, where $\tilde{I} \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds P_i' is valid, and $\tilde{x}_i$ is the secret corresponding to the witness w_i.
8. $b' \leftarrow D(\mathsf{out})$.

Lemma 4. *Assuming that the PVSS scheme is correct, the outputs of experiments* $\mathbf{H_3}$ *and* $\mathbf{H_4}$ *are indistinguishable.*

Proof. This is a series of hybrids, where we change the contribution of each honest dealer one-by-one. By the correctness property of the PVSS scheme all verifications checks in the protocol pass (for the secret distributed by this honest dealer) and the reconstructed secret is the same as the honest dealer shared during the sharing phase.

Hybrid $\mathbf{H_5}$: This hybrid is the same as before, except that the challenger stops its interaction with A after the sharing phase.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j'\in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P_i' \in H$:

(a) For each P_i such that $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs}) = 1$, check whether $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.

(b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\textsc{proof}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If

$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, w) \notin R,$$

then abort.

(c) For each valid P_i publish

$$(s_j^{(i)}, \textsc{proof}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}, \tau).$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk_i'\}_{i\in[2t+1]}, \textsc{proof}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in I\cap H} x_i \oplus \tilde{x}_{i\in\tilde{I}}$, where $\tilde{I} \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds P_i' is valid, and $\tilde{x}_i$ is the secret corresponding to the witness w_i.
8. $b' \leftarrow D(\mathsf{out})$.

Lemma 5. *The outputs of experiments* $\mathbf{H_4}$ *and* $\mathbf{H_5}$ *are indistinguishable.*

Proof. Note that in the previous hybrid the protocol output, which is exactly the input of the distinguisher D, was *already* fixed and could be computed by the challenger by the end of the sharing phase. Thus, nothing changed.

Hybrid $\mathbf{H_6}$: This hybrid is the same as before, except that in the beginning of the sharing phase each honest dealer P_i now chooses a value x_i' uniformly at random. Each encryption of a share sent by an honest dealer to a party P_j is now changed to an encryption of a corresponding share of x_i'.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j'\in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Sample $x_i' \leftarrow \{0,1\}$ uniformly at random.
   (c) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i', \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P_i' \in H$:

(a) For each P_i such that

$$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, \mathsf{crs}) = 1,$$

check whether $\textsc{proof}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by P_i is not the same as one distributed by any dealer P_k, where $k < i$. Denote P_i as *valid* if so.

(b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\textsc{proof}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If

$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \textsc{proof}_D^{(i)}, w) \notin R,$$

then abort.

6. $\mathsf{out} \leftarrow \bigoplus_{i\in I\cap H} x_i \oplus \tilde{x}_{i\in\tilde{I}}$, where $I' \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds P_i' is valid, and $\tilde{x}_i$ is the secret corresponding to the witness w_i.
7. $b' \leftarrow D(\mathsf{out})$.

Lemma 6. *Assuming that the PVSS scheme satisfies the IND1-secrecy property, the outputs of experiments* $\mathbf{H_6}$ *and* $\mathbf{H_7}$ *are computationally indistinguishable.*

Proof. This is a series of hybrids, where we change sets encryptions sent by each honest dealer D_i one-by-one. Given a PPT adversary A and a distinguisher D who is able to distinguish between the two hybrids given the output of the challenger's interaction with A, we construct an adversary B on the IND1-secrecy property of the underlying PVSS scheme as follows. Upon obtaining the PVSS setup information as well as the public keys of the honest parties from its challenger C, B generates new PVSS setup, and forwards this setup information to the adversary A, along with the public keys of the honest parties generated by C. Then, B forwards the public keys supplied by A to C. B follows the game as specified by the previous hybrid, except when it needs to act as the dealer D_i. Then, upon obtaining the ciphertexts, $\textsc{proof}_D$, and x_b from its challenger, B generates a new $\textsc{proof}_D'$ using the simulator for the setup generated by B, and forwards the ciphertexts along with the new $\textsc{proof}_D'$ to the adversary A. Then, B continues to follow the game as specified by the previous hybrid. When computing the output of the protocol, B uses x_b as the contribution from the honest dealer. B outputs exactly what D outputs. Note that if x_b was x_0, the game played corresponds exactly to the game in the previous hybrid. Otherwise, the game played corresponds exactly to the game specified in the new hybrid. Thus, if the advantage of the pair A and D is non-negligible, B's advantage is non-negligible as well.

Note that in the last hybrid the information about each honest secret x_i the adversary receives during the sharing phase of the protocol is completely independent of the honest secrets values x_i. This corresponds to the security game outlined in Definition 1 with the bit b fixed to 1.

C Execution Leaks with ElGamal Commitments

Protocol 3 EL Coin Tossing from ElGamal Commitments

Sharing phase:
Each D_i, $i \in [t+1]$ does the following:

1. D_i chooses random degree-t polynomials f_1 and f_2:

$$f_1 = a_0 + a_1 x + \cdots + a_t x^t,$$

$$f_2 = b_0 + b_1 x + \cdots + b_t x^t,$$

.

2. D_i chooses a pair of generators (g, h).
3. D_i commits to f_1 and f_2 via broadcasting (g, h) along with

$$(c_0, c_1, \cdots, c_t) = \Big((g^{a_0}, h^{a_0} \cdot g^{b_0}), (g^{a_1}, h^{a_1} \cdot g^{b_1}), \cdots, (g^{a_t}, h^{a_t} \cdot g^{b_t})\Big).$$

4. D_i sends $r_j = f_1(j)$ and $s_j = f_2(j)$ to each P_j, $j \in [2t+1]$; and sends polynomials f_1 and f_2 to D'_i.

Each R_i, $i \in [2t+1]$ does the following:

1. For each dealer D_j, R_i checks whether the share (r_i, s_i) it obtained from D_j, and the commitments to f_1 and f_2 distributed by D_j satisfy

$$g^{r_i} = \prod_{k=0}^{t} (g^{a_k})^{i^k},$$

$$h^{r_i} \cdot g^{s_i} = \prod_{k=0}^{t} \Big(h^{a_k} \cdot g^{b_k}\Big)^{i^k}$$

   If not, R_i broadcasts $\mathsf{Complain} - \mathsf{D_j}$.
2. R_i sends all shares (r_i, s_i) that passed verification to every R'_j.

Each D'_i, $i \in [t+1]$ does the following:

1. D'_i broadcasts shares of parties who complained about D_i. If any share broadcast by D'_i does not pass the check above, D'_i is deemed corrupt.

Reconstruction phase:
Each R'_i, $i \in [t+1]$ does the following:

1. If R_i complained about D_j, and D'_j was not deemed corrupt, R'_i sets the i-th corresponding share to s_i and r_i broadcast by D'_j.
2. R'_i outputs all shares (s_i, r_i) it obtained for non-corrupt dealers.

Client C does the following:

1. For each D'_i who was not deemed corrupt, C uses any $t+1$ shares s_j and r_j that pass the verification check against the corresponding commitment to reconstruct the value $s_i = f_i(0)$, where f_i is the polynomial f_2 dealt by D_i/D'_i.
2. Let H denote the index set of dealers D'_i which were not deemed corrupt. C outputs $\bigoplus_{i \in H} s_i$.

D Randomness Generation Performance per Role

We separately measure the performance of each role in our linearized VSS construction, which serves as a subprotocol in our randomness generation. Because of this, we do not report the times for the second role of the dealer D' (as there are no complains, D' is not required to do anything). We also do not report the time for the second role of each receiver R_i': These parties simply forward messages they received, which requires only very little time. We report our findings in the Table 3.

$(t, n = 5t + 4)$	Dealer D	Receiver R_i	Client
(1, 9)	25	26	54
(2, 14)	38	29	88
(4, 24)	63	35	177
(8, 44)	114	46	423

Table 3: Performance of each role in our SL Randomness Generation, all running times in milliseconds.

As expected, the dealing of the secret (work done by the Dealer), and the secret reconstruction (work done by the Client) are the two most expensive operations in our protocol.

We further report the communication sizes of each role below. Note that as Client is not required to publish anything, we do not report any numbers for them. Further, note that in the honest case, both Receiver roles publish exactly the same information.

$(t, n = 5t + 4)$	Dealer	Receiver
(1, 9)	0.00129	0.00006
(2, 14)	0.0018	0.00006
(4, 24)	0.0028	0.00006
(8, 44)	0.0048	0.00006

Table 4: Performance of each role in our SL Randomness Generation, communication sizes in MB.

As expected, the Dealer is the one who posts the most, and receiver's communication does not increase with n (note that we consider only a single VSS). For the dealer, the communication size increases linearly with t.