Towards Practical Committee Sizes in YOSO MPC

Paweł Kędzior[1], Marcin Mielniczuk[2], and Daniele Venturi[3]

[1] IDEAS Research Institute, Poland.
p.kedzior@mimuw.edu.pl
[2] University of Warsaw, Poland.
m.mielniczuk@uw.edu.pl
[3] Sapienza University of Rome, Italy.
venturi@di.uniroma1.it

Abstract. Secure multi-party computation in permissionless settings like blockchains faces the challenge of maintaining security under adaptive corruptions. YOSO MPC (Benhamouda *et al.*, TCC 2020; Gentry *et al.*, Crypto 2021) addresses this through ephemeral roles, allowing committee members to speak only once before erasing state, thus preventing adaptive attacks within each role's lifetime. However, prior work on YOSO MPC relied on an honest majority within committees and required enormous committee sizes, often numbering in the thousands.

In this paper, we challenge both requirements by leveraging the cryptoeconomic properties of the underlying blockchain. We introduce a mechanism for identifying parties who fail to fulfill their roles during protocol execution, using time-lock puzzles. This enables misbehaving parties to be penalized based on publicly verifiable fraud proofs, allowing for a more aggressive committee threshold. Furthermore, we propose a new sortition procedure that ensures the resulting committee always has a constant size, not just in expectation. This reduces fluctuations in committee size, enabling an even higher committee threshold. These techniques allow us to handle any constant fraction $f < 1$ of total corruptions.

Finally, we refine and generalize the analysis of corruptions in YOSO protocols, obtaining tighter bounds. Combined with our other enhancements, this enables committees of around 100 parties. Our analysis yields a 70–80% improvement over the estimates of the seminal work by Benhamouda *et al.* (TCC 2020). We believe our work paves the way for practical deployments of YOSO MPC protocols.

1 Introduction

Threshold protocols are the staple of modern cryptography, providing a way to balance opposing design requirements. A prominent example is the inherent tension between *secrecy* and *robustness* in t-out-of-n secret sharing. Ideally, we would like to set the threshold t as high as possible, possibly even $t = n$, to make sure that the secret remains protected even if the adversary corrupts many of the parties. However, this makes the protocol increasingly fragile, as with $t = n$ a failure of as little as one party makes the secret unrecoverable. This leads to the common practice of setting t to a value significantly lower than n, e.g., $t = n/2$. A similar tension occurs in the setting of multi-party computation (MPC), where the properties of *privacy* and *fairness* (or *guaranteed output delivery*) are in opposition.

On the other hand, there *is* an asymmetry between these two properties, especially in situations where it may be possible to *detect* a breach of robustness, whereas a breach of secrecy is a passive misbehavior that might inherently be undetectable. While detection of misbehavior does not prevent it from happening in the first place, it opens the door to its deterrence through cryptoeconomic incentives, such as punishments, as explored in, for instance, [ADMM14, BK14, KB14, DFLM24, BDF+25].

1.1 YOSO MPC and the honest-majority assumption

The framework of You-Only-Speak-Once (YOSO) MPC [BGG+20, GHK+21] recently emerged to exploit the potential of large-scale computation in permissionless settings. In YOSO MPC a participating machine

takes on a specific role only briefly: it receives messages, carries out a local computation (depending on the role), and transmits messages within a single communication round to the next committee of participating machines. Before transmitting its message, the machine deletes all other protocol-related state information. A crucial property of YOSO MPC is the fact that the adversary cannot predict the next committee of participating machines; this property enables the design of protocols with communication sub-linear in the total number of parties N, while the adversary can still corrupt up to T parties, where T is a constant fraction of N.

In more detail, YOSO MPC distinguishes between (stateful) physical machines and (stateless) abstract roles, which are deployed on demand to perform a certain task. An important step in the design of YOSO MPC is to select which physical machine is performing a given role. This can be modeled as a hybrid *role assignment* functionality, as formalized by Canetti *et al.* [CKR+23], leading to a modular protocol design in MPC frameworks with a composition theorem. The main idea behind role assignment [BGG+20, CKR+23] is to use two types of committees. The members of the *nominating* committee do not need to receive any private input and may therefore self–select through cryptographic sortition (as in Algorand [CM19]). Subsequently, each nominator is expected to nominate a random party to participate in the *holding* committee. More precisely, each nominator samples fresh ephemeral keys for the role it is nominating and broadcasts the public key along with an encryption of the secret key under anonymous public-key encryption. All other machines simply observe the broadcast channel and try to decrypt each ciphertext: in case of success, the machine knows it has been nominated and can obtain the messages sent to its prescribed role. The members of the holding committee can thereby receive input from the previous committee, fulfill the computation corresponding to their role, and send the output to the future committee.

A prominent limitation of the majority of YOSO MPC protocols is the assumption that each committee admits an honest majority of roles, which is problematic in protocols involving a large number of rounds, such as proactive secret sharing (PSS), where we would like to keep a secret for a long time. For example, an adversary who can obtain a dishonest majority in a single run with a mere probability of 1%, will succeed to obtain at least *one* majority over the course of 100 rounds with probability at least $1 - 0.99^{100} \approx 63\%$. This shows that even rare failures accumulate quickly, undermining long-term protocols like PSS. Furthermore, the honest majority requirement concerns abstract roles, not physical machines. Since the role assignment itself is subject to corruption, most prior works on YOSO only tolerate up to 29% of corruptions in the entire system [BGG+20, GHK+21, CKR+23]. We refer to Section 1.3 for a more detailed discussion of works that fall outside of the typical paradigm.

1.2 Our contributions

In this paper, we investigate strategies to mitigate the honest-majority assumption in YOSO MPC. Our main technical contributions are highlighted below.

Enhanced role assignment: identifiable aborts Our first contribution is the introduction of support for identifiable aborts in YOSO MPC. More precisely, we extend the ideal functionality $\mathcal{F}_{\mathsf{RA}}$ and the protocol Π_{RA} to support identifying parties who did not fulfill their assigned roles.

This stands in apparent contradiction with the unpredictability requirement of the committees; we resolve this conflict by requiring that the identities of the nominees remain hidden for at least δ rounds, where δ is a fixed constant.

We augment the role assignment protocol based on holding and nominating committees [BGG+20, CKR+23] by letting the nominators broadcast, along with the ciphertexts, a *time-lock puzzle* (TLP) [BDD+21a] storing the identity of the nominee; the TLP basically hides the identity of the nominated party for up to δ rounds, where δ is a fixed parameter, after which it can be opened to reveal the nominee's identity. This allows us to identify nominated parties that refused to speak. To obtain malicious security, we also include a non-interactive zero-knowledge (NIZK) proof showing that (1) the TLP contains a valid identity (2) the ephemeral decryption and signing keys correspond to the encryption and verification keys, respectively. Formalizing this idea poses a significant challenge, as in the plain UC framework, functionalities are not

meant to be able to access each other's internal states. To mitigate this problem, we adopt the *global UC* framework [CDPW07].

Finally, the modular design of [CKR+23] consists of the role assignment functionality and the compiler using the role assignment functionality to compile abstract protocols (for roles) into concrete protocols (for machines). We modify their compiler and obtain one that turns any abstract YOSO protocol *with identifiable aborts* into a concrete YOSO protocol *with identifiable aborts*, using our augmented functionality.

We emphasize that, although TLPs are computationally expensive to solve, they are intended to *never actually be solved* during the normal operation of the system. On the contrary, TLPs would only be solved in the event of an abort, and the threat of punishment would deter them in the first place. Such phenomena have already been observed in practice; for instance, not a single Arbitrum fraud proof was submitted from its launch in 2021 until 2023 [Fel23], which, to our knowledge, has not changed to this day.

Enhanced role assignment: sortition. Our third contribution is a new sortition procedure that ensures a constant committee size n in each round. Prior works have relied on a sortition procedure similar to the one used by Algorand [CM19], in which the nominating committee forms based on a VRF-based lottery. More precisely, each user evaluates a verifiable random function (VRF) on a common random input, and only those users whose VRF output is below a certain threshold are selected to form the nominating committee. This procedure inherently introduces variance in the committee size, and—looking ahead—this variance poses a significant problem when the committee size is chosen to be very small.

We introduce a two-stage sortition procedure, which may be of independent interest, that ensures that the committee size is *exactly* n,not only so in expectation. This procedure preserves the efficiency requirements of prior works, and more precisely, admits communication complexity linear in the committee size n (and independent of the universe size N). The main idea is to first select a *pre-committee* of *expected* size, say, $2n$ using the standard VRF-based lottery. Given that the *actual* committee size follows a binomial distribution with standard deviation close to $\sqrt{2n}$, it will, with overwhelming probability, contain at least n users. Furthermore, the distribution of corruptions in the pre-committee reflects the corruption rate in the overall system. Therefore, a random subset of the pre-committee of size n would inherit this property and be a good candidate for the nominating committee.

This subset, however, must be unpredictable to the adversary: for example, if the slots in the nominating committee were assigned on a first-come-first-served basis, a rushing adversary (who could delay the communication of the honest users) could easily obtain a disproportionate number of seats in the nominating committee; similarly if the adversary could predict which users have a disproportionately higher chance of being selected to the final committee, it could corrupt specifically these parties. Conversely, the unpredictability is also important to ensure the participation of users facing blockchain transaction fees for participation, who could refrain from submitting their candidacy if they could predict that they have a low chance of being selected to the final nominating committee. This is where randomness beacons come to the rescue. A randomness beacon is a trusted service that broadcasts fresh, unpredictable, and publicly verifiable random values at regular intervals; a heuristic example is the latest Bitcoin block hash. If parties have access to such a beacon, and nominations were done in round r, they can use the beacon value from round $r+1$ to select the final nominating committee.

It turns out that this simple change complicates the analysis of the committee composition, as the random variables "I have been selected" are no longer independent. Our major contribution is a tighter and more general analysis of the corruptions in the holding committee. We improve upon [BGG+20, GHK+21, CKR+23] providing a unified bound on the number Z of corruptions in the holding committee. By combining this with a variant of the Chernoff bound that is significantly tighter in the parameter region of interest, we obtain a much more accurate tail bounds for corruptions in the holding committee. Our main technical tool to this end is the use of the theory of *negative association*.

Dealing with aborts in the nominating committee. While our techniques based on time-lock puzzles address aborts in the holding committee, they do not foreclose the possibility of attacking the system by causing aborts in the nominating committee. This kind of aborts are extremely difficult to identify—aborts

in the nominating committee are *passive* misbehaviors and, from the outside, a party who chose not to speak cannot be distinguished from a party who did not have the right to speak in the first place.

However, as another contribution, our new sortition mechanism addresses this issue. The main idea behind deterring this kind of aborts is to ensure that this behavior only undermines the power of the adversary. In our two-stage sortition mechanisms, this is easy to achieve by setting the "overshoot", i.e., the expected committee size, to be large enough that the honest parties could fill the committee themselves should the adversary abort in the nomination phase. This means that the best strategy for the adversary is to submit the winning draws in the nomination phase, hoping it can break the secrecy, unable to abort in the holding committee without being traced (and penalized).

The countermeasure suffers from the clear drawback that, in the case of extremely high corruption rates f close to 1, it causes a significant fraction of the pre-committee members to be discarded and not selected for the final committee. Given that blockchain transactions come with a fee, this would undermine the incentives to participate in the system. As another contribution, we propose an alternative mechanism that could be combined with our new sortition mechanism to achieve the best of two worlds: to keep the cost of deterring the adversary low when the system is under its normal operation and only incur additional overhead if the system is under attack by a malicious party controlling a large fraction of the system. Furthermore, this additional overhead is not financial, but rather takes the form of a small (constant) latency.

The main idea is to *rerun* the sortition in a clever way. Suppose that instead of n parties only $(1-s)c$ were nominated, leaving the holding committee with a shortage of sc parties. In this case, one could run the sortition again with *the same* randomness but *different* (and disjoint) winning criteria. In particular, the winning probability would be adjusted based on the missing fraction s. For instance, if the initial sortition selected parties whose VRF output was in the interval $[0, 1000)$, and only the half of the committee was successfully nominated,[4] then the rerun would only select parties whose VRF output was in the interval $[1000, 1500)$. Crucially, all draws that were not used in a given run of the sortition would be forfeited and could not be reused in the reruns. This way, the adversary gains nothing by trying to sabotage the nominating committee, as it would simply lose its nominations, eventually leading to a holding committee with a *surplus* of honest parties. Moreover, the reruns would quickly converge, and the full committee is selected in a number of rounds that does not depend on n. It scales with the *actual* corruption rate f as $1/(1-f)$, whereas the previous approach scaled with the *maximal* corruption rate.

Relaxing the honest-majority assumption. Prior works on YOSO MPC relied solely on cryptographic techniques to ensure the security of the protocol. However, YOSO MPC is usually intended to be used in a blockchain setting, which often provides much more than just a communication channel and a consensus layer. Users of such systems own coins of real monetary value, which can be used to provide cryptoeconomic incentives to the participants of the protocol. In fact, permissionless protocols often rely on the rationality of the participants. For instance, many papers analyzing Bitcoin often justify its security on the rather unrealistic honest-majority assumption, but, as prior works have explored [BGM+18], Bitcoin merely requires that the miners are rational.

The techniques we have established can be used to relax the honest-majority assumption in YOSO MPC. Thanks to the support for identifying aborts and the malicious security of the underlying protocol, any misbehavior—including aborts—results in a fraud proof. Such fraud proofs are publicly verifiable and can be used to penalize the misbehaving parties and incentivize honest behavior. Note that this type of robustness is slightly weaker than the traditional one, as it relies on cryptoeconomic incentives rather than absolute cryptographic guarantees. However, looking ahead, these penalties turn out to be quite severe, plausibly discouraging any attempts to sabotage the robustness of the protocol. This justifies our technique of an aggressive choice of the threshold for the holding committee.

Concretely small committees. The motivation of YOSO MPC is practical in nature, and it stems from the observation that committee-based MPC is necessary to deploy MPC in practice in large-scale systems

[4] Note that, at this point, the long-term identities of the nominated parties are not yet known.

such as blockchains. This is due to the fact that the major bottleneck for typical MPC protocols is the communication overhead, which grows quadratically with the number of parties. Unfortunately, prior works [BGG+20] concluded that the committees must be enormous to ensure security, on the order of thousands of members. This order of magnitude would render many applications of YOSO-based MPC impractical.

As our final contribution, we challenge this long-standing requirement. We use our techniques to show that YOSO-based protocols can be secure with committees of modest sizes. The sizes we achieve are in the ballpark of 100, possibly smaller in some security regimes, larger in the most challenging of settings. Furthermore, our tighter tail bounds for corruptions in the holding committee let us decrease the committee sizes required by [BGG+20] by 70–80%. As a concrete example, our results show that a YOSO protocol can withstand a $f = 25\%$ corruption rate with a committee size of $n = 76$, relative threshold $\beta = 0.8$ (i.e., using βn-out-of-n secret sharing).

Furthermore, we explore the plausibility of ensuring robustness of YOSO protocols under the premise that YOSO were integrated with the Ethereum consensus protocol in a similar fashion to how Algorand consensus is based on YOSO. We consider the scenario in which the nodes of the network are Ethereum validators with their usual deposit of 32 ETH, and the penalties are incurred in a similar fashion to Ethereum's slashing mechanism, that is, as correlation penalties. Correlation penalties are meant to be forgiving towards accidental misbehaviors (such as temporary network outage), but harshly punish coordinated misbehaviors. Under the assumption that the penalty for the abort is proportional to the number of parties who have aborted in a given round, we compute the minimal cost of breaking the robustness of the protocol, showing that the penalties can indeed be a powerful tool for maintaining the security of YOSO-based protocols. Continuing the concrete example from the previous paragraph, we estimate that any adversary wishing to cause a robustness break in that scenario requires an expenditure of 352 ETH in penalties. While the quadratic communication complexity would still admittedly make general MPC a significant endeavor in that setting, we believe that our work paves the way towards practical deployments of YOSO-based protocols in various applications.

1.3 Related work

Following [BGG+20, GHK+21, CKR+23], several other works improved the design of MPC protocols in the YOSO framework along different dimensions [BHK+22, CDGK22, CDK+22, KRY24, CD24, LMR+24, BCR+24, LMR+25, DKLN25, EMP25]. Most of these works require that *each* committee admits an honest majority (often to obtain guaranteed output delivery), but this requirement has also been challenged: Bhadauria *et al.* [BCR+24] explore mitigating honest-majority in the YOSO line, yet still require the existence of at least *one* honest-majority committee that performs a function-independent computation to achieve guaranteed output delivery. Other works attempted to mitigate the "corruption-doubling" effect of two-stage sortition by the use of random-index PIR [GHM+21] and random-index ORAM [HK22]. Notably, the latter work is the only use known to us of the theory of negative association in the context of sortition and YOSO.

Beyond the literature focused on YOSO, the line of research closest to our contributions is *Le Mans* [RS22], which studies dynamic/maximally fluid MPC under dishonest majority. However, *Le Mans* provides only "security with abort" guarantees and does not come with *identifiable* aborts, providing no accountability for misbehavior. Our work targets exactly this gap: we strengthen YOSO role assignment to obtain *identifiable* aborts while preserving the anonymity of parties until they speak via delayed identification, and we build on this to obtain YOSO-style proactive secret sharing (YOSSO) tolerating arbitrary corruption fractions $f < 1$ *under fraud proofs*, that is, with publicly identifiable aborts.

Prevention of secrecy breaches under full corrupt has been studied by the line of works on *secret sharing with snitching* [DFLM24, BDF+25]. However, these works are only secure assuming the adversary consists of several mutually distrustful subadversaries, and do not allow for resharing of the secret in the style of PSS.

Other works have explored the use of time-lock puzzles to identify aborts in MPC [FHKS21, SSS21] with the goal of compiling passively secure MPC to covertly secure MPC. The work of [BDD+21a] achieves UC-secure two-party computation with *output independent* aborts using time-lock puzzles. However, none of these

works provide malicious security in the YOSO model while simultaneously preserving role unpredictability and enabling publicly verifiable identification of aborting nominees, which is the focus of this paper.

2 Preliminaries

We denote the security parameter by λ and we write $[n] = \{1, \ldots, n\}$. For a deterministic algorithm $\mathcal{A}$, we use $y := \mathcal{A}(x)$ to denote the output of deterministic algorithms. For a probabilistic algorithm $\mathcal{A}$, we write $y := \mathcal{A}(x; r)$ to denote its output with random coins r. We write $y \leftarrow \mathcal{A}(x)$ when the random coins r are implicitly assumed to be uniformly random, and we denote sampling uniformly from a set by $x \leftarrow\!\$\ X$. When we wish to ignore the output of an algorithm, we write $_ \leftarrow \mathcal{A}(x)$ or $(y, _) \leftarrow \mathcal{A}(x)$.

2.1 UC in the synchronous model

Our protocols rely on a round structure in which every honest machine is guaranteed an opportunity to *speak in each round*: while the adversary may be rushing within a round, it cannot delay an honest party's message beyond the end of that round. Concretely, we assume lock-step rounds and a known upper bound of one round on message delay, so any message sent by an honest party in round r is delivered (and hence available for processing) by the beginning of round $r+1$. This synchrony assumption is standard in the YOSO/role-assignment formalism [CKR+23] and can be modeled in the (G)UC framework by working in the synchronous setting of Katz *et al.* [KMTZ13].

2.2 Time-lock puzzles.

Time-lock puzzles (TLPs, sometimes called timed-release encryption) are one of the simplest time-based cryptographic primitives, which protect the secrecy of the message for a certain amount of time T. They allow the owner of a secret message m to lock it into a puzzle puz in time $t \ll T$. Anyone can try to solve the puzzle to recover the original message m, but it takes at least sequential time T to do so. Perhaps surprisingly, TLPs admit a UC-secure realization [BDD+21a], which we use as a building block in our construction.

The functionality $\mathcal{F}_{\mathsf{TLP}}$ of [BDD+21a] allows parties to create puzzles by sending $(\textsc{CreatePuzzle}, \mathsf{sid}, m, \Gamma)$ to the functionality, where m is the message to be locked and Γ is the number of rounds it takes to solve the puzzle, and responds with $(\textsc{CreatedPuzzle}, \mathsf{sid}, \mathsf{puz} = (\mathsf{st}_0, \Gamma, \mathtt{tag}))$, where st_0 is the initial state and $\mathtt{tag}$ uniquely identifies the emitted puzzle. Solving puzzles works as follows: first, one sends $(\textsc{Solve}, \mathsf{sid}, \mathsf{st})$ with the current state st of the puzzle to advance the computation. Since TLPs require sequential one, one can solve multiple of them in parallel, and after a round has elapsed, one sends $(\textsc{Output}, \mathsf{sid})$, to which the functionality responds with $(\textsc{Complete}, \mathsf{sid}, L_i)$ where L_i is an array of new states for all of them. Finally, once the TLP has reached its final state st^*, the locked message can be obtained by sending $(\textsc{GetMsg}, \mathsf{sid}, \mathsf{puz}, \mathsf{st}^*)$. We refer the reader to Appendix B for the formal ideal functionality $\mathcal{F}_{\mathsf{TLP}}$ from [BDD+21a].

2.3 Bulletin board

Similarly to [CKR+23], we adopt the formalism of a bulletin board $\mathcal{F}_{\mathsf{BB}}$, which can be instantiated by a blockchain and is modeled by an ideal functionality $\mathcal{F}_{\mathsf{BB}}$. One can write to $\mathcal{F}_{\mathsf{BB}}$ by sending a message $(\textsc{Write}, \mathsf{sid}, msg)$ and read messages published in round r by sending $(\textsc{Read}, \mathsf{sid}, r)$. We refer the reader to Appendix D for the formal definition of $\mathcal{F}_{\mathsf{BB}}$.

2.4 Properties of Negative Association

Our main technical tool for the analysis of the new sortition procedure is the theory of Negatively Associated (NA) random variables, introduced by Joag-Dev and Proschan [JDP83]. Negative association captures a form of "negative dependence" where if one subset of variables is large, the remaining variables tend to be small. Formally, a collection of random variables $Y_1, \ldots, Y_N$ is NA if for every pair of disjoint index

sets $I, J \subseteq \{1, \ldots, N\}$ and any coordinate-wise non-decreasing functions f, g, it holds that $\mathrm{Cov}(f(Y_i, i \in I), g(Y_j, j \in J)) \leq 0$.

The following closure properties make NA a powerful framework for analyzing sampling and randomized assignments [DP09, JDP83].

Lemma 1 (Closure under Subsets). *If $(Y_1, \ldots, Y_N)$ is NA, then any subset of these variables is also NA.*

Lemma 2 (Permutation Distributions). *Let $x_1, \ldots, x_N$ be any fixed real numbers. If π is a uniform random permutation of $\{1, \ldots, N\}$, then the random variables $X_1 = x_{\pi(1)}, \ldots, X_N = x_{\pi(N)}$ are Negatively Associated [JDP83].*

This implies that sampling n items without replacement from a population of size N yields NA indicator variables. Specifically, if we place n balls into N bins with capacity 1, the indicators $\{I_{i,j}\}$ where $I_{i,j} = 1$ iff ball i is in bin j, are NA.

Lemma 3 (Closure under Disjoint Monotone Transformations). *Let $\mathbf{Y} = (Y_1, \ldots, \ldots, Y_m)$ be a vector of NA random variables. Let $f_1, \ldots, f_k$ be non-decreasing functions defined on disjoint subsets of indices $S_1, \ldots, S_k \subseteq \{1, \ldots, m\}$. Then the random variables*

$$X_j = f_j(\{Y_i : i \in S_j\}), \quad j = 1, \ldots, k$$

are also NA.

Lemma 4 (Closure under Independent Products). *If $\mathbf{X}$ and $\mathbf{Y}$ are two independent vectors of NA random variables, then the concatenated vector $(\mathbf{X}, \mathbf{Y})$ is also NA.*

A powerful property of NA variables is that they satisfy the same Chernoff bounds as sums of independent random variables (this follows from the Moment Generating Function domination properties). Given that the tightest form of the Chernoff bound is formulated in terms of the Kullback-Leibler divergence (a.k.a. relative entropy), we state:

Theorem 1 (Chernoff Bound for NA Variables). *Let $X_1, \ldots, X_n$ be NA random variables taking values in $[0, 1]$. Let $Z = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[Z]$. For any $t > \mu$:*

$$\Pr[Z \geq t] \leq \exp\left(-D\left(\frac{t}{n} \middle\| \frac{\mu}{n}\right) n\right),$$

where

$$D(p\|q) = p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q}$$

is the Kullback-Leibler divergence. A more practical (but slightly weaker) bound, obtained via standard simplifications [DP09], is:

$$\Pr[Z \geq (1+\delta)\mu] \leq \exp\left(-\frac{\delta^2 \mu}{2+\delta}\right).$$

3 Identifying aborts in YOSO MPC

One of our main building blocks for our enhanced role assignment is the support for identifying aborts in holding committees. At a high level, we achieve it by relying on time-lock puzzles (TLPs) containing the long-term identities of the nominees. The puzzles are configured in such a way that they hide the identities long enough for the holding committee to perform its task. This approach only mitigates the aborts in the holding committee; aborts in the nominating committee are prevented in a separate way which is discussed in Section 4.

In this section, we present how to formalize this notion using an ideal functionality, leading to our enhanced role assignment functionality $\mathcal{F}_{\mathsf{ERA}}$. We then present a protocol Π_{ERA} realizing $\mathcal{F}_{\mathsf{ERA}}$. The security is jointly proved for all of our contributions in Section 6.

All of these contributions build upon the role assignment of [CKR+23]. Given that the original figures of [CKR+23] are quite lengthy, we only present the modifications in the full body of the paper and defer the complete figures to Appendices E to G.

3.1 Ideal functionality

Our enhanced role assignment functionality $\mathcal{F}_{\mathsf{ERA}}$ is similar to the role assignment functionality $\mathcal{F}_{\mathsf{RA}}$ of [CKR+23]. It assigns machines to computation roles while keeping this assignment hidden until the point at which the machines actually speak.[5] Our first enhancement provided in $\mathcal{F}_{\mathsf{ERA}}$ is ensuring that parties who were nominated and did not speak in their assigned turn can be identified. This stands in apparent contradiction, and is reconciled by the requirement that the identities of the nominees remain hidden for at least δ rounds, for a suitably chosen constant δ. We require that δ is strictly larger than the number of rounds `delay` needed to complete nominations. Looking ahead, this will be guaranteed by the use of time-lock puzzles.

More precisely, our functionality $\mathcal{F}_{\mathsf{ERA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{D}, \mathtt{delay}, \delta)$ extends the role assignment functionality $\mathcal{F}_{\mathsf{RA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{D}, \mathtt{delay})$ of [CKR+23]. It takes an additional parameter δ, as discussed above:

> $\delta \geq$ `delay`: the minimum number of rounds for which the identities of nominees remain hidden.

In addition to the commands of $\mathcal{F}_{\mathsf{RA}}$, we add the command IdentifyNominee, which waits for δ rounds and returns the identifier pid of the nominee who was assigned the ephemeral public key ek:

> **Identify nominee:** On input $(\textsc{IdentifyNominee}, \mathsf{cid}, r, \mathsf{ek})$ from M in round $r' \geq r + 1$, send $(\textsc{IdentifyNominee}, \mathsf{cid}, r, \mathsf{ek})$ to $\mathcal{S}$ and wait until round $r' + \delta$ and continue accepting commands in parallel.
>
> In round $r' + \delta$, send $(\textsc{IdentifyNominee}, \mathsf{cid}, r, \mathsf{ek}, \mathsf{pid})$ to $\mathcal{S}$ and M, where pid is the unique value such that $(\mathsf{ek}, \mathsf{vk}, R) \in \mathsf{PKeys}$ and $(\mathsf{pid}, _, _, R) \in \mathsf{SKeys}$, with $\mathsf{ek}, \mathsf{vk} \neq \bot$.

Note that this already allows one to trace the role R who did not speak in its turn, as the PKeys field contains publicly accessible tuples $(\mathsf{ek}, \mathsf{vk}, R)$. The condition $ek, vk \neq \bot$ is purely technical and maintains compatibility with the original design of functionality $\mathcal{F}_{\mathsf{RA}}$, which pads the nominee public key list PKeys with dummy entries $(\mathsf{pid}, \bot, \bot)$ where pid belongs to an honest party. We opted to keep compatibility with the original functionality $\mathcal{F}_{\mathsf{RA}}$ to keep the changes to the functionality minimal.

Identification of nominees is, however, not enough to guarantee the safety of the system. One way the adversary could attack the system is to nominate an honest party P_i, but assign it a decryption key dk which does not match the ephemeral public key ek. This way the party would have no option but to abort, being unable to decrypt any incoming messages. To prevent this attack, the functionality will refuse to register keys that do not form a valid key pair. Formally, we modify the command Nom-Corrupt, by adding the following check *before* registering the keys but *after* increasing the corruption counters:

> Verify that $(ek, dk) \neq (\bot, \bot)$ and $(vk, sk) \neq (\bot, \bot)$ indeed form key pairs. If any of the checks fails, return.

Looking ahead, this will be ensured by the use of NIZKs. Note that such invalid nominations still count towards the corruption limit, since — in the motivating case where nominators self-select through cryptographic sortition — the adversary can only spend the winning sortition draw once.

All in all, the functionality has a significant overlap with the functionality presented in [CKR+23], and we defer the complete ideal functionality $\mathcal{F}_{\mathsf{ERA}}$ to Appendix E.

[5] Note that which machines provide input to the computation – and receive output from the computation – could be determined in some fixed, external way, depending on the application; therefore we consider only the assignment of machines to computation roles, and not input and output roles.

The value of the parameter δ. We do not specify the actual value of δ, as it might depend on the protocol using $\mathcal{F}_{\mathsf{ERA}}$. In particular, it is important that the identities of the nominees remain hidden until they get to speak. If the parties need several rounds to complete their part of the protocol, then δ might be much larger than $\mathtt{delay}$. On the other hand, if the parties can speak immediately after being nominated, then a small δ, such as $\delta = \mathtt{delay} + 1$, might be a good choice. This will, looking ahead, be the case in our YOSSO protocol of Section 7.

The validity of secret keys. We chose to ensure the validity of the secret keys that the nominees receive at the level of functionality. Note that the functionality actually identifies aborts, but only nominees, and identifying aborts is the responsibility of the layer using $\mathcal{F}_{\mathsf{ERA}}$. A different design choice would be to let nominees prove in ZK that the secret keys they received were invalid. This, however, would put a significant burden on the honest parties.

3.2 Protocol

Just as in the case of the ideal functionality, we build upon the protocol of [CKR+23], and we augment the role assignment protocol Π_{RA} of [CKR+23] as follows.

The nomination process, which is a part of the command New originally provided by Π_{RA}, is extended in the following way: after the ephemeral keys for the nominee have been encrypted, a time-lock puzzle storing the identity of the nominee pid is created together with a NIZK for the relation $\mathcal{R}_{\mathsf{nom}}$ specified Appendix I.1, proving that (i) the TLP contains a valid $\mathsf{pid} \in \mathcal{P}$, (ii) the verification key vk corresponds to the signing key sk, and (iii) the encryption key ek corresponds to the decryption key dk. Formally, we add the following steps to the nomination process:

- Send $(\textsc{CreatePuzzle}, \mathsf{cid}, \delta, \mathsf{pid})$ to $\mathcal{F}_{\mathsf{TLP}}$.
- On $(\textsc{CreatedPuzzle}, \mathsf{cid}, \mathsf{tlp})$ from $\mathcal{F}_{\mathsf{TLP}}$, let $x = (\mathsf{ek}, \mathsf{vk}, \mathsf{setP}, \mathsf{ct}, \mathsf{tlp}, \delta)$ and $w = (\mathsf{dk}, \mathsf{sk}, j, P_j.\mathsf{pk}, r_{\mathsf{ct}})$, where r_{ct} is the randomness used to create ct. Send $(\textsc{Prove}, \mathsf{cid}, \mathsf{ssid}, x, w)$ to $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$.

In the end, the TLP together with the NIZK are included in the bulletin board record. Moreover, as the parties learn their assigned roles through the command Read originally provided by Π_{RA}, they normally check the validity of the sortition draws. We additionally let them ignore any nominations that fail the validity checks captured by the relation $\mathcal{R}_{\mathsf{nom}}$:

For the remaining elements, let $x_j = (\mathsf{ek}, \mathsf{vk}, \mathsf{setP}, \mathsf{ct}_j, \mathsf{tlp}_j, \delta)$ and send $(\textsc{Verify}, \mathsf{cid}, \mathsf{ssid}, x_j, \pi_j^{\mathsf{nom}})$ to $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$. Remove the element from the list if $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$ rejects, i.e., sends $(\textsc{Verification}, \mathsf{cid}, \mathsf{ssid}, \mathbf{false})$.

Finally, the identification command IdentifyNominee is implemented through solving the TLPs that were generated during the nominations:

Identify Nominee: Party P_i on input $(\textsc{IdentifyNominee}, \mathsf{cid}, r, \mathsf{ek})$ from the environment $\mathcal{Z}$ in round $r' \geq r+1$, send $\textsc{Read}(\mathsf{bid}, r)$ to $\mathcal{F}_{\mathsf{BB}}$ and retrieve an entry of the form $(\mathsf{bid}, r, \mathsf{ek}, \mathsf{vk}, c, \mathsf{draw}, \pi, \mathsf{tlp}, \pi^{\mathsf{nom}})$. If no such entry exists, return $\perp$ immediately. Parse $\mathsf{tlp} =: (\mathsf{st}_0, \Gamma, \mathtt{tag})$.

1. Set $\mathsf{st} := \mathsf{st}_0$.
2. **For** $t = 1$ **to** δ **do**
   (a) Send $(\textsc{Solve}, \mathsf{sid}, \mathsf{st})$ to $\mathcal{F}_{\mathsf{TLP}}$.
   (b) Wait until the next round, i.e., round $r' + t$.
   (c) Send $(\textsc{Output}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{TLP}}$ and receive $(\textsc{Complete}, \mathsf{sid}, L_i)$.
   (d) If there exists $(\mathsf{P}_i, \mathsf{sid}, \mathsf{st}, \mathsf{st}') \in L_i$, then set $\mathsf{st} := \mathsf{st}'$.
3. Send $(\textsc{GetMsg}, \mathsf{sid}, \mathsf{st}_0, \mathsf{tlp}, \mathsf{st})$ to $\mathcal{F}_{\mathsf{TLP}}$.

4. On $(\textsc{GetMsg}, \mathsf{sid}, \mathsf{st}_0, tag, \mathsf{st}, \mathsf{pid})$ with $\mathsf{pid} \neq \perp$ and **output** pid to everyone in round $r' + \delta$.

Since the functionality has a significant overlap with the functionality presented in [CKR+23], we defer the complete protocol Π_{ERA} to Appendix F.

Remarks

Authentication of messages to $\mathcal{F}_{\mathsf{BB}}$. We do not formally model the authentication performed by the bulletin board, but we note that one could enforce that the only parties who are allowed to perform the nominations are ones who hold some stake on the blockchain. This would prevent an adversary from a Sybil attack allowing use of multiple VRF keys for the sortition to increase the chance of being nominated.

NIZK proofs for the TLPs. Our functionality includes a zero-knowledge proof that the TLP contains a valid party identifier. One way to instantiate them could be to use a TLP scheme based on repeated squaring based on RSA groups together with RSA accumulators, leading to a NIZK implemented natively in the RSA group. Given that the creator of the NIZK knows the trapdoor for the TLP, we expect these NIZKs to be quite efficient.

However, we note that this requirement could be removed if we were to assume that the nomination is performed using nominating committees, as in the case of Π_{ERA}. In this case, any nominator who submits a TLP that does *not* contain a valid party identifier could be punished by the judge instead of the nominee.

Why GUC? The nomination-validity check $\mathcal{R}_{\mathsf{nom}}$ must certify, among other things, that the published time-lock puzzle tlp indeed commits to a valid nominee identity. Since we model time-locking via the ideal functionality $\mathcal{F}_{\mathsf{TLP}}$ (rather than a concrete NP-checkable puzzle scheme), this check is *not* a standalone NP relation over the public transcript: it requires consulting the internal state of $\mathcal{F}_{\mathsf{TLP}}$. However, UC functionalities typically cannot peek into the internal state of other functionalities. To keep our approach rigorous, we therefore work in the Global-UC framework, treating $\mathcal{F}_{\mathsf{TLP}}$ and $\mathcal{F}_{\mathsf{NIZK}}$ as global resources that can enforce such cross-functionality invariants. Accordingly, writing $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$ is an abuse of notation: $\mathcal{R}_{\mathsf{nom}}$ should be read as a shorthand for this joint-state predicate rather than a plain NP relation. See Appendix I for details.

Security. The security is jointly proved for all of our contributions in Section 6.

4 Enhanced sortition procedure

The second building block of our enhanced role assignment procedure is the two-stage nomination procedure which guarantees an exact committee size. This property distinguishes it from the typical cryptographic sortition, which only achieves the desired committee size n in expectation. and not only achieves it in expectation. We believe that this contribution could be of independent interest.

As outlined in the introduction, in order to obtain committee assignment, YOSO relies on nominating committees, which self-select through cryptographic sortition. In cryptographic sortition a la Algorand [CM19], each party P_i privately evaluates a Verifiable Random Function (VRF) on some common random input. If the output y_i is below a specific threshold T, the party is "selected" and obtains the right to perform an action, e.g., nominate a committee member. Because the selection is independent across parties, the total number of winners follows a binomial distribution $\mathsf{Bin}(N, s)$, only achieving a certain committee size n in expectation. Crucially, the sortition keeps the communication low, as only $\Theta(n)$ parties need to send messages to the blockchain, independently of the total number of parties N.

An important detail of the role assignment functionality $\mathcal{F}_{\mathsf{RA}}$ of [CKR+23] is the use of the $\perp$ values to "pad" nominations. More precisely, the functionality syntactically considers that the committee is always of fixed size n, however, it might be the case that some committee members are not identifiable. It happens when the adversary sends empty keys $(\mathsf{ek}, \mathsf{vk}, R) = (\perp, \perp, \perp)$ for his pool of corrupt nominations. In actual

instantiations based on cryptographic sortition, such as Π_{RA}, the authors choose the committee size n so that no nominations are ever discarded, and so the expected number of nominations is *strictly fewer* than n. In the case of Π_{RA}, this choice of n is necessary to prevent the adversary from biasing the committee composition. Consider the following case: let the nominations in Π_{RA} be c' greater than n, but then only the top n are selected e.g., by sorting by the ephemeral encryption key ek, and then choosing only the top n elemets. Encryption key ek would be random for honest nominators, but could be easily biased for dishonest nominations, for example by rejection sampling in until the first 3 bits of ek are zero. This would easily let the adversary ensure that only honest nominations are discarded.

One could naively argue that this is only a matter of choosing a committee size on the order of the security parameter λ. While it is indeed true that prior works [BGG+20, GHK+21, CKR+23] assume such committee sizes to ensure tail bounds for breaches of secrecy and liveness, the devil is in the dependence on the admissible corruption threshold f-or, equivalently, the holding committee threshold β. In fact, [BGG+20] noticed that the size of the committee needs to scale as $\Omega(\lambda/(1-\beta))$. Such scaling would hamper our main technique of aggressively choosing the holding committee threshold β. More concretely, suppose that the committee of expected size n performs proactive secret sharing with threshold $t = \beta n$. Using standard role assignment, the actual committee size is a random variable with the distribution $\mathsf{Bin}(N, s)$ where $n = Ns$, and admits the standard deviation $\sigma = \sqrt{Ns(1-s)} \approx \sqrt{n}$, which remains large *regardless* of how large the total number of parties N is set to be. As a rule of a thumb, the actual committee size will be in the range $n \pm 3\sigma \approx n \pm 3\sqrt{n}$. This means that if we choose $\beta/n \approx 3\sqrt{n}$, then with noticeable probability the committee will be too small to hand the torch even under a *completely honest execution*. As a concrete example, consider a committee of expected size $n = 128$, in which case the standard deviation is $\sigma \approx 11$. Then, with $\beta \approx 95/128 = 0.7422$, the next committee might turn out to be too small to proceed with the next round. Note that $\beta = 0.7422$ corresponds to $f \approx 0.492$. The situation only becomes more dire if we want to achieve practical committee sizes (which we, looking ahead, do in Section 5). While the analysis above is only a rule of thumb, it shows that the variance of the sortition procedure becomes an issue if we want to both handle high corruption thresholds and achieve practical committee sizes.

Our main idea is to use a two-stage sortition procedure to obtain a committee of *exactly* size n. The first stage involves a standard cryptographic sortition aimed to obtain a pre-committee of expected size $m > n$, which overwhelmingly contains at least n members. In the second stage, we choose a subset of size n from the pre-committee. More concretely, consider the following strawman sortition procedure, where n is the expected committee size:

- Each party P_i evaluates the VRF privately, obtaining output $y_i \in [0, 1]$.
- If $y_i < 2C/N$, then P_i submits (i, y_i, π_i) to the bulletin board.
- After the deadline has passed, the committee is formed by taking the n parties with the lowest values of y_i. If less than n parties submitted their draws, the sortition aborts.

Unfortunately, in practice, bulletin board is a blockchain, and submissions come with a transaction fee. If $y_i \approx 2C/N$, then even if y_i is a winning draw, it is unlikely to be selected in the second round, and therefore the party P_i would lose money paying the transaction fee and would likely not submit its value to the bulletin board at all. Without assumptions on the incentives for participating in the sortition and the transaction fees, it would not even be possible to analyze the security of this procedure. To circumvent this issue, we require that the second stage chooses a *random* subset of size n from the pre-committee. More concretely, one orders the parties in the pre-committee according to their draws y_i, permutes them with a random permutation ϕderived from the randomness beacon value of the next round, and chooses the top n parties in the permuted ordering. In fact, at this moment it is easy to notice that the actual ordering of the parties in the pre-committee is actually irrelevant given that the permutation ϕ is uniformly random. Finally, we observe that this modification does not affect the YOSO property of the sortition, as long as the nominators submit their nomination along with their winning draw, as it happens in the original protocol. The only difference now is that some pre-nominations will be discarded. The full sortition procedure is described below, for an appropriate constant α. We remark that our protocol abstracts from the incentives of the participants, as the actual game theoretic analysis is considered out of scope for this work. The final sortition is presented in Figure 1. We defer integration with the Role Assignment to Section 4.1.

Two-stage sortition with fixed committee size

1. In round R, each party P_i evaluates the VRF privately, obtaining output y_i.
2. If $y_i < \alpha C/N$, then P_i submits (i, y_i, π_i, m) to the bulletin board where m is the nomination message.
3. Let S be the set of all parties that submitted a valid message in step 2. In round $R+1$, the beacon value is used to choose a random subset $A \subset S$ such that $|A| = n$.
4. The committee is formed by taking the parties in A and their nominations are honored. The remaining nominations are discarded.

Fig. 1: Our generic two-stage sortition protocol, guaranteeing a fixed committee size. The protocol depends on a verifiable random function (VRF) and a randomness beacon. The value $\alpha > 1$ is an appropriate constant.

4.1 Integration with Role Assignment

In order to integrate this sortition procedure with the existing role assignment functionality and protocol, we observe that the protocol Π_{RA} already performs a step that is very similar at the *syntactical* level. More specifically, the protocol Π_{RA} uses a permutation ϕ of the nominations, however, it serves a different purpose. In their protocol, no nominations are ever discarded, and the permutation is used to ensure that the *roles* assigned to committee members are unpredictable, which is crucial for the security of the abstract-to-concrete compiler. In our case, the permutation is used to ensure that one cannot predict the actual committee *members*.

It turns out that the changes to the functionality introduced in Section 3.1 *already* guarantee that the committee contains *exactly* n members. To see why, notice that the Nom-Corrupt already requires that each ephemeral keypair is valid, and $\perp$ values are no longer permitted.

Remark 1. In Section 3, we have already discussed how to provide identifiable aborts in the holding committee through the use of time-lock puzzles. However, an adversary could use a much more insidious technique to cause a protocol abort: simply refrain from submitting its winning draw to the bulletin board. This means that the approach of Section 3 would only *halve* the number of aborts that an adversarially controlled roles could cause. The use of our new sortition procedure solves this issue as well, by adjusting the winning probability for the pre-committee by a factor of $1/(1-f)$. We describe a more fine-grained solution in Section 4.3.

Security. The security is jointly proved for all of our contributions in Section 6.

4.2 Analysis

In this section, we analyze the security of the above sortition procedure.

Probabilistic model Our two-stage sortition procedure selects a committee of *exactly* size n from the N parties. We wish to analyze the number of corrupt parties in the committee, given that fN-out-of-N parties are corrupt. For the purpose of analysis, consider the strawman sortition described before, which can be modeled as follows (assuming no aborts): each party P_i randomly selects a value $X_i \in \{0,1\}^\lambda$. The committee is formed by the n parties P_i who obtained the smallest values of X_i.

Since the values X_i are drawn uniformly and independently from a large domain $\{0,1\}^\lambda$, the probability of any collision is negligible in security parameter (formally: $\Pr[\exists i \neq j : X_i = X_j] \leq N^2/2^{\lambda+1}$). Conditioned on all X_i being distinct, the values $X_1, \ldots, X_N$ induce a random permutation of the parties. By symmetry, any subset of N parties of size n is equally likely to be the set of parties having the n smallest values. Consequently, this process is statistically close to selecting n parties uniformly at random from the set of all N parties without replacement. Therefore, the number of corrupt parties Y in the nominating committee follows a hypergeometric distribution:

$$Y \sim \mathsf{Hypergeometric}(N, fN, n).$$

Finally, notice that permuting the collected candidates with a random permutation (as done in our actual protocol to select the final n members from the pre-committee) does not skew this distribution. Since the permutation is derived from a beacon value unpredictable at the time of VRF evaluation, it is independent of the set of corrupt parties. Thus, selecting the top n candidates after a random permutation is equivalent to selecting a random subset of size n, maintaining the hypergeometric distribution of corrupt *nominating* committee members. Our next goal remains to analyze the number Z of corrupt parties in the *holding* committee.

Tail bounds. A key ingredient in our approach to sortition is the observation that it satisfies tail bounds comparable to those of the standard sortition procedure. Our main technical tool is the theory of negative association (NA) random variables (cf. Section 2.4), and our analysis also carries over to the usual sortition procedure, simplifying the analysis of [BGG+20, CKR+23].

Balls-and-bins model. To establish the connection of our sortition procedure to negative association, we first rephrase the sortition procedure in the language of balls and bins, which is a common framework for analyzing randomized algorithms. Let $\mathcal{K} \subset [N]$ be the set of corrupt parties, with $|\mathcal{K}| = fN$. The sortition can be rephrased in the balls-and-bins model as putting n balls (committee seats) into N bins (parties participating in the sortition) without replacement. Let $I_{i,j}$ be the indicator of the event "ball i was assigned to bin j", which corresponds to the event "party j was assigned to seat i of the nominating committee". As per Section 2.4, it is known that the variables $\{I_{i,j}\}_{i,j}$ are NA. Furthermore, we can write $Y = \sum_{i=1}^{n} C_i$ where $C_i = 1$ if the i-th nominator is corrupt, that is $C_i = \sum_{j \in \mathcal{K}} I_{i,j}$. Since C_i is a non-decreasing function of the indicators $\{I_{i,j}\}_j$, by Lemma 3, the family $\{C_i\}$ is also NA.

Finally, we can write $Z = \sum_{i=1}^{n} X_i$ where X_i is the indicator that the i-th role of the holding committee is corrupt, Assuming that corrupt nominators always nominate corrupt parties and honest nominators nominate corrupt parties with probability f, we can write $X_i = C_i + (1 - C_i)B_i$. Since both C_i and B_i take values in $\{0, 1\}$, it holds that $X_i = \max(C_i, B_i)$. Again, the variable X_i is a non-decreasing function of B_i and C_i and by Lemma 3, the family $\{X_i\}_{i \in [n]}$ is NA. Therefore, we can invoke Theorem 1 to obtain a tail bound on Z. Crucially, the same bound is obtained in the usual sortition, in which the distribution of Y is binomial (and not hypergeometric), and the indicators C_i are independent instead of negatively associated. Intuitively, it is the case because the hypergeometric distribution is more concentrated than the binomial distribution. We stress that, unlike [BGG+20], we do not need to separate bound the number of corrupt nominators and the number of corrupt nominees chosen by honest nominators.

Corruptions in the holding committee. For concreteness, we spell out the tail bounds for the number of corrupt holding committee member Z, as they will be necessary for our security analysis. Notice that since B_i is independent of C_i, the probability that the role i in corrupted equals $\mathbb{E}[X_i] = \mathbb{E}[C_i] + (1 - \mathbb{E}[C_i])f = 2f - f^2 = f(2 - f) =: p^*$, and so $\mathbb{E}[Z] = np^*$. Then, for $\beta > p^*$, we can phrase the Chernoff bound in the KL form for Z:

$$\Pr[Z \geq \beta n] \leq \exp\left(-nD(\beta \| p^*)\right) \tag{1}$$

To instantiate the simpler bound, we take $1 + \delta = \beta/p^*$, i.e., $\delta = (\beta - p^*)/p^*$, obtaining

$$\Pr[Z \geq \beta n] \leq \exp\left(-n\frac{(\beta - p^*)^2 p^*}{\beta + p^*}\right) \tag{2}$$

Both $D(\beta \| p^*)$ and $(\beta - p^*)^2$ vanish at $\beta = p^*$, which is the boundary of the applicability of the bound. That is, the bound becomes trivial as β approaches p^*. This corresponds to the phenomenon of "doubling" the corruption budget. In particular, for any constant choice of $\beta > p^*$, the probability bounded by either Equations (1) and (2) is negligible in n.

We do not present tail bounds for the number of corrupt nominators Y, however, we note that the techniques of negative association easily lead to tail bounds for Y.

Security proof is deferred to Section 6.

4.3 Robust nomination with sequential eligibility windows

Recall that an adversary can silently attack the system already at the nomination layer by withholding its winning sortition submissions. Our basic two-stage sortition already addresses it by "overshooting" by a multiplicative factor of $1/(1-f)$ to make sure that, even if all corrupt lottery winners abort, the honest parties alone can still fill the committee on their own; in other words, the two-stage sortition sets the expected pre-committee size to $m = (1+\eta)/(1-f)n$. Then, the honest parties alone still contribute $\mathbb{E}[X] = (1-f)m = (1+\eta)n$ nominations on average, and at least n nominations with overwhelming probability. When the final committee of size exactly n is selected, the procedure discards a fraction

$$1 - (1-f)/(1+\eta) = \frac{f+\eta}{1+\eta}$$

of all submissions, which approaches 1 as f approaches 1. For moderate values of f, such overhead is acceptable, but for extremely high corruption rate f close to 1 the required overshoot $1/(1-f)$ may become so large that it undermines honest parties' incentives to participate in the system in the first place, and causes a denial-of-service on cryptoeconomic grounds. More importantly, in real-life deployments, the corruption factor f is not the *actual* fraction of corrupt parties, but rather an upper bound on it. This means that merely a *threat* of an adversary corrupting a vast majority of the parties would be sufficient to sabotage the system, without the adversary actually corrupting anyone.

We outline how to mitigate this issue. Our design goal is to ensure that the honest users of the system only pay a small overhead when the system is under normal operation (less than f' misbehavior rate), and only pay an extra cost when the system is under active attack (withstanding any adversary corrupting less that f fraction of the parties). Furthermore, this extra cost is not financial, but rather a small *latency* overhead.

Resilient rerun sortition. Fix two thresholds $0 < f' < f < 1$, where f is the *maximal* supported corruption rate and f' represents a "normal-operation" level corresponding to accidental/non-coordinated misbehavior. Our main idea is to keep the initial pre-committee sortition calibrated to the corruption rate f', but allow reruns that gradually unlock fresh *disjoint* winning regions of the VRF output space under the *same* beacon value. At a high level, given that the latency incurred by withholding the winning draws in not significant, the adversary does not benefit from doing so, as it only forfeits the winning draws and allows for a committee with an overrepresentation of the honest parties.

More precisely, let r be the nomination round, and let ρ be the randomness beacon value for round r, used to derive VRF outputs in the sortition. For each party P_i, let $y_i \in [0,1)$ denote its (deterministic) VRF output on the common input $(r, \rho, \ldots)$. We *adaptively* partition the interval $[0,1)$ into disjoint windows $[T_0, T_1), [T_1, T_2), \ldots$ with $T_0 = 0$. In run $j \geq 1$, a party is eligible to submit *only if* $y_i \in [T_{j-1}, T_j)$, and so any draw can be used in at most one rerun.

We proceed iteratively. Let n_j be the number of nominations submitted in runs $1, 2, \ldots, j$, and let $n_0 \coloneqq 0$. In the run $j = 1, 2, \ldots$ we set the window length so that the *target* expected number of total winners (under the optimistic threshold f') is

$$m_j \coloneqq \frac{(1+\eta)(n - n_{j-1})}{1-f'}.$$

Equivalently, set $T_j := T_{j-1} + m_j/N$. The procedure halts if $n_j \geq n$. At this point, we form the final committee of size n by applying the beacon-derived random permutation to *all* collected submissions and taking the first n. Note that approach is egalitarian in the sense that parties from the subsequent windows have equal incentives to the ones who win in the first window.

Latency/discard–rate tradeoff. Intuitively, if the actual corruption is f (and not f'), and consequently only a $(1-f)$ fraction of lottery winners submit their winning draws, then in each rerun the expected number of newly filled seats is a constant fraction of the remaining deficit:

$$\mathbb{E}[n_j - n_{j-1}] \;\approx\; (1-f)m_j \;=\; (1+\eta)\frac{1-f}{1-f'}(n - n_{j-1}).$$

Thus the deficit shrinks geometrically, and the number of reruns required is $O(\log n)$ for fixed f', f, η; in practice for n in the hundreds this is a small constant. Moreover, unlike the simple one–shot procedure based on overshooting calibrated to f, the scheme increases latency (incurs extra rounds) *only when* a shortage beyond the normal rate is observed.

Given that the changes to the functionality, protocol, and the security proof are straightforward and require heavy bookkeeping, we only outline the changes to avoid notational overload, and forego a fully formal model for this .

Changes to $\mathcal{F}_{\mathsf{ERA}}$ and Π_{ERA}. This rerun mechanism affects only the nomination *delay*. Formally, set the *worst-case* nomination bound $\mathtt{delay}_{\mathsf{rr}}$ to cover the maximum number of reruns we allow (chosen so that failure is negligible), and require $\delta \geq \mathtt{delay}_{\mathsf{rr}}$ in $\mathcal{F}_{\mathsf{ERA}}$. In Π_{ERA}, the NEW procedure is extended so that nominations may be posted in rerun rounds $r < r' < r + \mathtt{delay}_{\mathsf{rr}}$, each tagged with its rerun index j, as long as the rerun turns out to be necessary. This means that the protocol needs to keep track of the previous runs to be able to appropriately choose the eligibility windows and the appropriate beacon value. The READ procedure collects all nominations for the same (cid, r) across these rounds before applying the beacon permutation and selecting the final n. Finally, the $\mathsf{Eligible}$ predicate required by $\mathcal{F}_{\mathsf{AS}}$ is extended to include all the additional information necessary to handle reruns.

Security. As previously mentioned, the security of our scheme is jointly proved for all of our contributions in Section 6.

5 Concrete analysis

In the previous section, we have abstractly established that the new two-stage sortition procedure can be incorporated into the role assignment procedure, and that the protocol implements the ideal functionality of interest. These results are asymptotic in nature, and operate under the assumption that the committee size is asymptotically large. This requirement stands in contrast with the practical need for small committees in real-world applications in order to minimize the communication overhead and latency of the protocol. Practical implementations of MPC typically face a prohibitive communication overhead when the number of participants reaches several hundreds, as the communication complexity of the protocols grows quadratically with the number of parties. The seminal work of [BGG+20] analyzes the minimal committee size required to retain secrecy depending on the corruption rate f. Unfortunately, the required committee size is on the order of thousands of parties. This means that each handover would involve posting millions of messages to the bulletin board, incurring large transaction costs and blockchain congestion.

In this section, we apply our techniques to obtain concretely small committees in YOSO MPC, small enough to be used drop-in with Ethereum committees of validators. Our high-level approach is the following. Let Z be the number of corrupt parties in the holding committee of size n. Given a statistical security parameter Δ (e.g., $\Delta = 30$), we aim to find the minimal committee size n that achieves $\Pr[Z \geq \beta n] < 2^{-\Delta}$. This inequality is clearly easier to satisfy for larger values of β. Therefore, in order to allow for a smaller committee size n, we can aggressively choose the relative threshold β, relying on the identifiable aborts (and penalties) to preserve robustness. However, very high values of β could lead to a spontaneous abort of the protocol due to the variance in the size of the holding committee. This is why we rely on the new sortition procedure introduced in Section 4.

5.1 Concretely small committees

To find the minimal committee size n providing Δ bits of statistical security, consider a relative threshold $\beta \in [1/2, 1)$ for the holding committee. In Section 4.2, we have established that the number of corrupted committee members Z can be modeled as a sum of negatively associated random variables, $Z = \sum_{i=1}^{n} X_i$, where $\mathbb{E}[X_i] = p^* = f(2 - f)$. Given a holding committee threshold $\beta > p^*$, this allows us to apply the Chernoff bound in the KL form:

$$\Pr[Z \geq \beta n] \leq \exp\left(-n \cdot D(\beta \| p^*)\right), \tag{3}$$

where $D(\beta\|p^*) = \beta \log \frac{\beta}{p^*} + (1-\beta) \log \frac{1-\beta}{1-p^*}$ is the Kullback-Leibler divergence between Bernoulli distributions with parameters β and p^*. To ensure that a secrecy violation occurs with probability at most $2^{-\Delta}$, we require:

$$\exp\left(-n \cdot D(\beta\|p^*)\right) \leq 2^{-\Delta}, \tag{4}$$

which leads to the following lower bound on the committee size n:

$$n \geq \left\lceil \frac{\Delta \log 2}{D(\beta\|p^*)} \right\rceil. \tag{5}$$

The resulting values of n for various choices of f and β are depicted in Figure 2, in the case of $\Delta = 30$. To allow for a comparison with the previous work of [BGG+20], we note that one can obtain the values for $\Delta = 128$ by multiplying the committee size by a factor of 4.25; for completeness we also present the values for $\Delta = 128$.

$f\backslash\beta$	0.50	0.60	0.70	0.80	0.90	0.95	0.98
0.01	9/17	8/13	7/10	7/8	7/7	6/6	6/6
0.05	20/40	17/28	14/20	13/16	11/12	11/11	10/10
0.10	43/86	32/52	24/34	20/24	17/18	15/15	14/14
0.15	95/189	56/92	39/55	29/36	23/25	20/21	18/18
0.20	255/510	106/176	62/88	42/52	30/33	26/27	23/23
0.25	1321/2641	236/392	105/149	61/76	40/44	33/34	29/29
0.30	∞	764/1273	196/280	92/115	54/60	42/44	37/37
0.40	∞	∞	1813/2589	274/342	106/117	73/76	57/58
0.50	∞	∞	∞	2376/2970	259/287	138/145	98/99
0.60	∞	∞	∞	∞	1241/1378	337/354	187/190
0.70	∞	∞	∞	∞	∞	1722/1812	480/489
0.80	∞	∞	∞	∞	∞	∞	3213/3278

(a) $\Delta = 30$

$f\backslash\beta$	0.50	0.60	0.70	0.80	0.90	0.95	0.98
0.01	35/70	32/53	30/42	28/34	26/28	25/26	24/24
0.05	85/170	71/117	60/85	52/65	45/50	42/44	41/41
0.10	183/366	131/218	102/145	82/102	68/75	61/64	57/58
0.15	403/805	236/393	163/232	121/151	94/104	82/86	76/77
0.20	1087/2174	450/749	262/373	176/219	126/139	107/112	97/98
0.25	5634/11268	1002/1670	443/632	258/322	169/187	138/145	121/123
0.30	∞	3260/5432	835/1192	393/491	227/252	178/187	152/155
0.40	∞	∞	7733/11047	1165/1456	448/497	305/321	243/247
0.50	∞	∞	∞	10137/12671	1103/1225	586/616	412/420
0.60	∞	∞	∞	∞	5292/5879	1436/1511	795/811
0.70	∞	∞	∞	∞	∞	7345/7731	2045/2086
0.80	∞	∞	∞	∞	∞	∞	13707/13986

(b) $\Delta = 128$

Fig. 2: Each entry t/n denotes the minimal committee size n and threshold $t = \lceil \beta n \rceil$ to achieve $\Pr[Z \geq \beta n] < 2^{-\Delta}$ for various corruption fractions f and secrecy thresholds β, and $\Delta = 30$.

5.2 Comparison with prior work

As it can be seen from the first column of Figure 2, our analysis provides a significant improvement over the committee sizes proposed by [BGG+20], decreasing the committee size by approximately 70–80%. This reduction stems from two primary technical refinements.

First, we replace the two-stage union bound approach in [BGG+20] with a single-stage concentration analysis of the total corrupted fraction. Specifically, the authors of [BGG+20] analyze corruptions in the holding committee as a sum of:

Direct Corruption (B_1) : The number of corrupted parties self-selected into the nominating committee via sortition, fn seats in expectation.

Indirect Corruption (B_2) : The number of corrupted parties nominated by honest members of the nominating committee, additional $f(1-f)n$ seats in expectation.

Our unified analysis avoids the "leakage" of the union bound, leading to a tighter estimate. Second, we utilize the tightest possible variant of the Chernoff bound based on the Kullback-Leibler divergence rather than the simplified bounds employed [BGG+20]. Conceptually, the standard Chernoff bound is obtained by Taylor expanding the KL divergence around $\beta \approx p^* \approx 0$. In their setting, $\beta = 0.5$ and $p^* < 0.5$, and the use of the tighter bound provides a significant improvement, regardless of the corruption threshold f. Finally, we observe that the choice of $\Delta = 128$ employed by [BGG+20] is a very conservative one, and typical for computational, and not statistical security. Given that the breaches at this level are only statistical, one can safely pick a lower value of Δ such as $\Delta = 30$.

The bounds used by prior work become even looser, as one increases β beyond 0.5, which is our main technique for reducing the committee size. This is why our analysis of Section 4.2 and the use of tighter bounds is crucial for obtaining our results of Figure 2.

5.3 Concrete attack costs

Since the liveness of the protocol is only guaranteed based on punishment and financial incentives, one needs to guarantee that they severe enough to actually deter misbehavior. In this section, we explore the hypothetical scenario in which the Ethereum blockchain adopts YOSO protocols for some of the validator duties, inheriting the penalty philosophy of their consensus mechanism. As a consequence, the honesty of the validators would be backed by the large deposits required by Ethereum, which are typically 32 ETH. This scenario shares similarities with the use of YOSO protocols by Algorand, however, we note that such YOSO committees could well be disjoint from the attestation committees used in the consensus scheme. One application could be to temporarily store secrets which are important for the functioning of the network. More concretely, a typical way to prevent Miner Extractable Value (MEV) attacks is to let users commit to their transactions and secret-share the underlying transaction contents with a committee. If the network uses a YOSO committees for proactive secret sharing of these transactions, a powerful adversary could attempt to trigger an abort of this protocol, causing the transaction data to be lost, effectively leading to a double spend.

The Ethereum slashing mechanism is specifically designed to penalize correlated failures more severely than isolated ones, given that coordinated attacks are much more damaging than isolated accidental misbehaviors. More precisely, validators incur correlation penalties and forfeit a fraction of the deposit proportional to the fraction of the total stake that misbehaved within a certain time window, such that each misbehaving validator only loses a small fraction of their deposit if they are the only party to be caught, and loses its whole deposit if 1/3 of the stake behaves maliciously.

Unlike the Ethereum consensus layer, which can tolerate the loss of liveness for several epochs and subsequently recover, proactive secret sharing (or YOSO MPC) remains vulnerable even if only one committee aborts. This is why we adopt a simplified model of this correlation penalty, in which all validators are weighed equally and the correlation penalty is proportional to the fraction of the committee that failed in a *single round*. Specifically, we assume each committee member stakes a deposit of d (e.g., $d = 32$ ETH). If A parties in a committee of size n abort simultaneously, the protocol imposes a correlation penalty $\rho = \min(A/n, 1) \cdot d$

on each cheating party. This means that the total cost of the attack scales quadratically with the number of aborts. A liveness attack succeeds $n - A < t$, or equivalently $A \geq n - t + 1$, and to analyze the minimal cost, we consider the case where $A = n - t + 1$.

We present the estimated attack costs in Figure 3 as *multiples of the deposit d*. For the purpose of this analysis, we adopt the committee sizes from Figure 2 (for $\Delta = 30$ and $\Delta = 128$). In other words, to obtain the cost in ETH, one should multiply the cost from the table by 32 ETH. The results indicate that due to the quadratic scaling of the correlation penalty, the economic cost of breaking liveness remains substantial even for smaller committees.

$f \backslash \beta$	0.50	0.60	0.70	0.80	0.90	0.95	0.98
0.01	9.00	6.00	4.00	1.50	0.43	0.50	0.50
0.05	21.00	12.00	7.00	3.00	1.00	0.27	0.30
0.10	44.00	21.00	10.68	3.12	0.67	0.20	0.21
0.15	95.00	37.00	15.76	5.33	1.08	0.57	0.17
0.20	256.00	71.00	24.85	6.98	1.45	0.44	0.13
0.25	1321.00	157.00	40.77	10.11	1.70	0.35	0.10
0.30	-	510.00	77.41	15.03	2.45	0.61	0.08
0.40	-	-	699.57	41.76	3.69	0.63	0.21
0.50	-	-	-	357.60	8.79	1.32	0.12
0.60	-	-	-	-	41.46	2.75	0.25
0.70	-	-	-	-	-	13.71	0.61
0.80	-	-	-	-	-	-	3.99

(a) $\Delta = 30$

$f \backslash \beta$	0.50	0.60	0.70	0.80	0.90	0.95	0.98
0.01	36.00	22.00	12.07	4.32	0.96	0.46	0.12
0.05	86.00	47.00	23.86	9.05	2.16	0.61	0.07
0.10	184.00	88.00	40.06	12.97	2.56	0.75	0.21
0.15	403.00	158.00	63.36	19.09	3.49	0.87	0.16
0.20	1088.00	300.00	100.89	26.52	4.23	0.96	0.12
0.25	5635.00	669.00	171.36	39.36	5.79	1.32	0.22
0.30	-	2173.00	322.56	59.88	8.05	1.60	0.31
0.40	-	-	2984.31	175.68	15.09	2.70	0.30
0.50	-	-	-	1521.48	37.05	4.68	0.58
0.60	-	-	-	-	176.43	11.47	1.07
0.70	-	-	-	-	-	58.12	2.54
0.80	-	-	-	-	-	-	16.82

(b) $\Delta = 128$

Fig. 3: Minimal liveness attack cost (in multiples of the deposit d) for committee sizes from Figure 2, under a correlation penalty proportional to the fraction of the committee that fails.

6 Security proofs

In Sections 3 and 4, we have introduced our enhancement to the role assignment procedure, formalizing it via the ideal functionality $\mathcal{F}_{\mathsf{ERA}}$ and presenting a protocol Π_{ERA}. We now proceed with a formal security proof.

6.1 UC security of Π_{ERA} (Theorem 2).

In this section, we state the precise theorem regarding the security of our protocol Π_{ERA} and outline the security proof.

To clarify the theorem statement, we recall that the construction of [CKR+23], which our work builds upon, employs *key-and-message non-committing encryption* as the anonymous encryption scheme used to publish ciphertexts containing nominations. At a high level, the non-committing property allows the simulator to equivocate on the content of t encrypted nominations if nominees are corrupted *before* speaking but *after* the nomination has been published.

KM-NCE admits a (parametrized) notion of CCA-security, denoted KMNC_k-CCA security, and an additional property called *unique recipient security* (KM-NCE-UR). Since our protocol's security proof uses KM-NCE in exactly the same way as [CKR+23] and the modifications that we introduce in the proof do not involve KM-NCE or any of its security properties, we omit the definitions and refer the reader to [CKR+23] for further details.

Theorem 2. *Let $\mathcal{T}(f) = n\left(1-(1-\varepsilon)(1-f)^2\right)$ be the threshold function, let $\delta \geq 3$, and let $\mathcal{U}$ be the uniform distribution. Suppose the KM-NCE scheme used has KMNC_k-CCA security (for $k = \mathrm{poly}(\kappa)$ being the bound on the roles fulfilled by a single machine) and KM-NCE-UR security, and the $\mathsf{Eligible}(r, \mathtt{tag})$ predicate returns 1 with probability*

$$s = \frac{1+\eta}{1-f} \cdot \frac{n}{N}$$

over the choice of r, for some constant $\eta > 0$.

Then, the protocol Π_{ERA} UC-realizes the functionality $\mathcal{F}_{\mathsf{ERA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{U}, 2, \delta)$ in the presence of $t \leq N$ adaptive corruptions in the $(\mathcal{F}_{\mathsf{Beacon}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{TLP}}, \mathcal{F}_{\mathsf{AS}})$-hybrid model.

We prove that Π_{ERA} (G)UC-realizes $\mathcal{F}_{\mathsf{ERA}}$ in the $(\mathcal{F}_{\mathsf{Beacon}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{NIZK}}[R_{\mathsf{nom}}], \mathcal{F}_{\mathsf{TLP}}, \mathcal{F}_{\mathsf{AS}})$-hybrid model by extending the simulator of [CKR+23, Theorem 7] to handle (i) time-lock encodings of nominee identities and (ii) nomination validity proofs tied to $\mathcal{F}_{\mathsf{TLP}}$. Concretely, the new simulator $\mathcal{S}_{\mathsf{ERA}}$ must ensure that the adversary learns nothing about a nominee's identity before the δ-round delay, while still being able to answer $\mathcal{F}_{\mathsf{ERA}}$.IdentifyNominee consistently after δ rounds. To this end, $\mathcal{S}_{\mathsf{ERA}}$ relies on a TLP idealization that provides a concealed setup trapdoor: when an honest nomination is created, $\mathcal{S}_{\mathsf{ERA}}$ produces a puzzle header that is indistinguishable from an honestly generated one but is initially bound to a dummy payload pid_0; once $\mathcal{F}_{\mathsf{ERA}}$ later fixes the true nominee identifier pid, $\mathcal{S}_{\mathsf{ERA}}$ uses the trapdoor to equivocate and rebind the already-posted puzzle so that any legitimate solve yields pid (and not pid_0), keeping the transcript consistent throughout. Furthermore, we change the way in which the simulator $\mathcal{S}_{\mathsf{ERA}}$ issues nomination calls to the functionality (Nom-Corrupt, Nom-Honest). This is because they should only be sent for members of the final committee, but not the pre-committee. Therefore, $\mathcal{S}_{\mathsf{ERA}}$ maintains a list of pre-committee members, selects a random subset of size n from it, and only issues nomination calls for those that end up in the final committee.

As in [CKR+23], the proof proceeds via a hybrid sequence from the real execution to the ideal execution with $\mathcal{S}_{\mathsf{ERA}}$. The only new steps are the hybrids that "factor out" the dependence on the concrete TLP interaction while preserving the delayed-opening semantics: first, we move to a hybrid where every honest nomination calls $\mathcal{F}_{\mathsf{TLP}}$.CreatePuzzle on a fixed dummy message $m_D := 0^{|m|}$ (hence making the puzzle creation message-independent) while $\mathcal{S}_{\mathsf{ERA}}$ records the real pid internally and later programs the δ-delayed opening to release that pid; indistinguishability up to round $r+\delta$ follows by reduction to the hiding property of TLPs up to delay δ, and after $r+\delta$ both worlds reveal the same pid. Next, we eliminate concrete puzzle creation entirely by replacing honestly generated puzzles with synthetic headers (distributed as fresh TLP headers) and using $\mathcal{F}_{\mathsf{TLP}}$ only as the delayed store that releases pid at time δ; since the previous hybrid already made honest puzzle headers message-independent, this replacement leaves the adversary's view unchanged.

Finally, we account for the nomination-validity layer by using $\mathcal{F}_{\mathsf{NIZK}}[R_{\mathsf{nom}}]$ in the standard simulation/extraction pattern: $\mathcal{S}_{\mathsf{ERA}}$ obtains simulated proofs for honest nominations, and for corrupted nominations it extracts a witness for π_{nom} and forwards the extracted nomination information to $\mathcal{F}_{\mathsf{ERA}}$, guaranteeing that accepted nominations are well-formed and that IdentifyNominee later identifies the committed nominee.

All in all, this yields indistinguishability between the real execution and the simulated world. The remaining hybrids then follow the same high-level framework as [CKR+23, Theorem 7] for the role-assignment layer (e.g., handling KM-NCE and bookkeeping around committee reads), and we do not modify those steps.

The only substantive deviation concerns the analysis of committee selection under adaptive corruptions. In [CKR+23] this is handled via a dedicated hybrid bounding the probability that the adversary "wins" the sortition more often than expected. In our case, we instead work in an $\mathcal{F}_{\mathsf{AS}}$-hybrid world (where eligibility is sampled ideally), so that the resulting committee distributions match exactly the probabilistic model analyzed in Section 4.2. We reintroduce the required bounds by applying the corresponding tail bounds for the number of corruptions in the holding committee, simplifying the analysis of [CKR+23].

Finally, we introduce a hybrid that aborts if the number of nominations is lower than n, which would contradict our aim of always choosing a committee of size exactly n. To argue that this abort only happens with negligible probability, we make sure that the expected size of the pre-committee is large enough that the honest parties can fill the committee on their own, should the adversary withhold all winning draws. Concretely, we increase the winning probability of the pre-committee sortition by a factor of $1/(1-f)$, leading to $s = (1+\eta)/(1-f)\cdot c/N$. This ensures that the expected number of honest winners is at least $(1+\eta)n$, and the claim follows by a standard Chernoff bound argument. Consequently, while the simulator-and-hybrids skeleton is inherited from [CKR+23], the sortition step is justified by our new two-stage sortition analysis.

Due to space limitations, the full simulator construction is deferred to Appendix G and the detailed proof of indistinguishability is deferred to Appendix H.

□

6.2 Robust nomination with sequential eligibility windows (Section 4.3).

We argue that the above modification preserves the security guarantees of the two-stage sortition and prevents nomination-layer aborts.

(1) Replacing VRFs by an idealized selection functionality. In our newest construction we do not use VRFs at all: we work directly in an $\mathcal{F}_{\mathsf{AS}}$-hybrid world, and we parameterize $\mathcal{F}_{\mathsf{AS}}$ with an *Eligible* predicate that implements the sequential eligibility windows.

Concretely, for each committee instance (identified by (r, ρ, cid)) and each party P_i, the predicate defines a single underlying draw $y_i \leftarrow\!\!\$\ [0,1)$ (sampled once and then reused for all reruns of that same instance). In rerun j, P_i is eligible to submit iff $y_i \in [T_{j-1}, T_j)$. Equivalently, we craft the tags given to $\mathcal{F}_{\mathsf{AS}}$ so that the eligibility check is uniquely tied to (r, ρ, cid) while the rerun index j selects the corresponding window, ensuring that any given y_i can justify eligibility in at most one rerun.

(2) Liveness of nominations under withholding. Let X_j be the number of *honest* winners in rerun j. Conditioned on the window boundaries, X_j is a sum of negatively associated indicator variables (and, in fact, follows the hypergeometric distribution characterizing sampling without replacement), and thus satisfies Chernoff-style concentration Theorem 1. Choosing $\mathtt{delay}_{\mathsf{rr}} = O(\log n)$ reruns implies that the probability that the cumulative honest submissions stay below n is negligible in n (and hence in λ for $n = \omega(\log \lambda)$), even if all corrupt winners abort.

(3) Committee composition. Note that, for each window, the nominee corruption rate is the same as in the original two-stage sortition, or better (if the adversary withheld the winning draw). Therefore, forming the final committee by choosing a random subset of size n will keep the bounds on the number of corrupt nominees intact.

□

7 Applying our results end-to-end

In our paper so far, we have focused on the particular components of a YOSO-style MPC protocol, such as the role assignment, the sortition procedure, and the concrete parameters of the role assignment. In this section, we outline how to apply these technique to a complete, end-to-end YOSO-style MPC protocol.

7.1 Abstract-to-natural compiler.

Following the blueprint of [CKR+23], we wish to ensure that any protocol in the abstract world (where the parties are abstract roles) can be securely implemented in the natural world (where the parties are concrete machines) in the presence of adaptive corruptions. In [CKR+23], this is achieved by a compiler which compiles any abstract-world YOSO protocol Π into a natural-world YOSO protocol $\mathsf{Compile}(\Pi)$. We modify the extended compiler of [CKR+23, Section 6], letting it handle the identification of misbehaving parties, and obtain the result that, if that the abstract protocol Π supports (publicly) identifiable aborts for roles itself, then the resulting natural-world protocol $\mathsf{Compile}(\Pi)$ supports (publicly) identifiable aborts for machines as well.

We defer the formal compiler statement and construction to Appendix K.

7.2 Proactive secret sharing

Given the extreme results of Section 7.1 are quite abstract and difficult to intuitively grasp, we illustrate the applications of our contributions with a YOSO-style proactive secret sharing scheme, which we call *YOSSO*. Our scheme improves upon the Evolving Committee Proactive Secret Sharing (ECPSS) scheme of [BGG+20], whose major limitations is the requirement of honest majority in the holding committee, which means that it can only tolerate an $f < 0.29$ fraction of corrupt parties in the system. While one could technically increase the reconstruction threshold t in the holding committee to allow for a higher fraction of corruptions, this would make the system unacceptably prone to denial-of-service attacks. The introduction of identifiable aborts allows us to circumvent this limitation and tolerate an arbitrary $f < 1$ fraction of corruptions.

To understand it, let us first recall the ECPSS scheme of [BGG+20] adapted to our notation. In their protocol, the committee nominated in round i has size c_i. Therefore, in each round, the parties present in the system check the bulletin board using the functionality $\mathcal{F}_{\mathsf{BB}}$ if they have been nominated, decrypt the messages sent to them. At the point when the size of the next committee c_{i+1} is already known, they compute a t-out-of-c_{i+1} resharing of the secret and send it to the next committee. In order to provide malicious security of the protocol, parties also provide NIZK proofs for the correctness of the resharing (cf. Appendix I).

We extend their design as follows. In the original scheme of [BGG+20], the dealer could be assumed to be honest. However, since in our case aborts are punishable, it is crucial that honest parties do not have to abort due to malformed messages they received, or—if they do—they should be able to publicly prove that the messages were malformed. To this end, we also ask the dealer to provide a NIZK proof of correctness of the initial sharing. Moreover, if the parties do not receive enough valid messages from the previous committee, they call a forensic procedure IdentifyNominee that identifies the parties that did not fulfill their role and output the set of misbehaving parties. We present the protocol's security analysis in Appendix J due to space limitations.

7.3 Deployment considerations

In this section, we briefly discuss how to implement the enforcement of penalties for YOSO protocols such as YOSSO. A natural solution is to use a *judge* smart contract, similarly as in [DFLM24, BDF+25]. The parties are be expected to deposit a collateral at protocol's outset, and the judge contract is supposed to slash the deposit in case of misbehavior. For simplicity, we will assume that the judge has access to $\mathcal{F}_{\mathsf{BB}}$, though this could be relaxed by routing nominations through the judge contract.

Smart contracts need to be lightweight, and to make sure that whistleblowing does not incur excessive transaction fees, one needs to avoid solving them on-chain. To this end, one can rely on publicly verifiable TLPs (see, for example, [BDD+20]). The party identifiers pid would then be chosen as the long-term public keys of the parties. In this case, judge receives as input the following data:

1. the time-lock puzzle tlp to be solved,
2. the solution pid^* together with a proof π_{sol} of its correctness

Upon such a submission, the judge then goes on to find the bulletin board record $(\mathsf{cid}, ek, vk, c, \mathsf{draw}, \pi, \mathsf{tlp}, \pi^{\mathsf{nom}})$ and verifies that tlp indeed corresponds to a valid nomination, i.e.:

1. (draw, π) is correct with respect to the VRF verification key vk,
2. $(\mathsf{tlp}, \pi^{\mathsf{nom}})$ passes the NIZK verification for the relation $\mathcal{R}_{\mathsf{nom}}$,
3. $(\mathsf{tlp}, \mathsf{pid}^*, \pi_{\mathsf{sol}})$ passes the verification of the time-lock puzzle solution

If all these checks pass, the judge punishes the party with the long-term public key pid^*, say, by slashing its deposit.

YOSSO Protocol Π^t_{YOSSO} in $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{ERA}}, \mathcal{F}_{\mathsf{TLP}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid world

Notation. Fix threshold parameters $t \leq n$. bid denotes the bulletin-board/blockchain instance; round r uses the subsession identifier $\mathsf{ssid} = (\mathsf{sid}, r)$ and the committee identifier $\mathsf{cid} = (\mathsf{bid}, r)$. The roles R are integers in $[n]$.

Round $r = 0$ (initial sharing). Upon input $(\textsc{Init}, \mathsf{sid}, \mathsf{bid}, s)$, the dealer does:

1. Send $(\textsc{Read}, \mathsf{cid}, 0)$ to $\mathcal{F}_{\mathsf{ERA}}$ and receive $(\textsc{Read}, \mathsf{cid}, 0, \mathsf{PKeys}, \mathsf{SKeys})$; parse $(\mathsf{ek}_1, \ldots, \mathsf{ek}_n)$ from PKeys.
2. Compute $(u_1, \ldots, u_n) \leftarrow \mathsf{Share}(s)$.
3. For each $k \in [n]$, let $e_k \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_k, u_k)$; let $\overline{\mathsf{e}} = (e_1, \ldots, e_n)$.
4. Define x and w according to $\mathcal{R}_{\mathsf{YOSSO}}$. Send $(\textsc{Prove}, \mathsf{sid}, \mathsf{ssid}, x, w)$ to $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{YOSSO}}]$ and receive $(\textsc{Proof}, \mathsf{sid}, \mathsf{ssid}, \pi)$.
5. Send $(\textsc{Write}, \mathsf{bid}, \mathsf{ssid}, \mathsf{e}, \pi)$ to $\mathcal{F}_{\mathsf{BB}}$.

Round $r > 0$. Upon input $(\textsc{Reshare}, \mathsf{sid}, \mathsf{bid}, r)$ party P_i does:

1. Send $(\textsc{Read}, \mathsf{cid}, r)$ to $\mathcal{F}_{\mathsf{ERA}}$ and receive $(\textsc{Read}, \mathsf{cid}, r, \mathsf{PKeys}, \mathsf{SKeys})$. If P_i is not nominated for round r, i.e., $\mathsf{SKeys} = \bot$, then halt. For each $(\mathsf{dk}, \mathsf{sk}, R) \in \mathsf{SKeys}$, call **Execute-Role**$(R, \mathsf{dk})$.

Execute-Role(R, dk):

1. Send $(\textsc{Read}, \mathsf{bid}, (\mathsf{sid}, r))$ to $\mathcal{F}_{\mathsf{BB}}$ and receive $\{(\textsc{Read}, \mathsf{bid}, \mathsf{ssid}, \overline{\mathsf{e}}_k, \pi_k)\}_{k \in [n]}$.
2. For $k \in [n]$, verify each π_k by sending $(\textsc{Verify}, \mathsf{cid}, \mathsf{ssid}, x_k, \pi_k)$, with x_k as per $\mathcal{R}_{\mathsf{YOSSO}}$ based on records from $\mathcal{F}_{\mathsf{BB}}$ for for rounds $r-1$ and r. Receive $(\textsc{Verification}, \mathsf{cid}, \mathsf{ssid}, b_k)$.
3. Let $S = \{k \in [n] \colon b_k = \mathbf{true}\}$. If $|S| < t$, then let $A \coloneqq [n] \setminus S$ and call $\textsc{IdentifyNominee}(A, \mathsf{PKeys})$ below.
4. Let $\mathsf{ct}_k \coloneqq \overline{\mathsf{e}}_k[R]$ be the ciphertext addressed at role R. Compute $u_k \coloneqq \mathsf{PKE.Dec}(\mathsf{dk}, \mathsf{ct}_k)$ for $k \in S$.
5. Reconstruct the secret: $u \coloneqq \mathsf{Reconstr}\left(\{(k, u_k)\}_{k \in S}\right)$.
6. Send $(\textsc{Read}, \mathsf{cid}, r{+}1)$ to $\mathcal{F}_{\mathsf{ERA}}$ and receive $(\textsc{Read}, \mathsf{cid}, r{+}1, \mathsf{PKeys}', _)$; parse $(\mathsf{ek}'_k)_{k \in [n]}$ from PKeys'.
7. Compute $(u'_1, \ldots, u'_n) \leftarrow \mathsf{Share}(u)$; set $e'_k \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}'_k, u'_k)$ and let $\overline{\mathsf{e}}' = (e'_1, \ldots, e'_n)$.
8. Prove correctness of the resharing: send $(\textsc{Prove}, \mathsf{sid}, \mathsf{ssid}', x, w)$ to $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{YOSSO}}]$ with x, w as per $\mathcal{R}_{\mathsf{YOSSO}}$. Wait to receive $(\textsc{Proof}, \mathsf{sid}, \mathsf{ssid}', \pi)$.
9. Send $(\textsc{Write}, \mathsf{bid}, \mathsf{ssid}', R, \overline{\mathsf{e}}', \pi)$ to $\mathcal{F}_{\mathsf{BB}}$.

Reconstruction. Each party P_i sends the share to the dealer.

IdentifyNominee(A, PKeys): For each $i \in A$, do:

1. Find the record $(\mathsf{ek}, \mathsf{vk}, i)$ in PKeys, if none exists, continue to the next i.
2. Send $(\textsc{IdentifyNominee}, \mathsf{cid}, r, ek)$ to $\mathcal{F}_{\mathsf{ERA}}$ and receive $(\textsc{IdentifyNominee}, \mathsf{cid}, \mathsf{pid})$. Record this value in the set P

Return the set P of parties who misbehaved in round r and halt.

7.4 Remarks on the protocol

Choice of cid. In our protocol, we set $\mathsf{cid} = (\mathsf{bid}, r)$, i.e., the committee is identified by the blockchain instance and the round number. This means that the committee is responsible for *all* secrets shared in round r on

blockchain bid. An alternative would be to set $\mathsf{cid} = (\mathsf{bid}, r, \mathsf{sid})$, where sid would identify a particular sharing instance or a particular user sharing the secret.

Use of the signature key. We note that our protocol does *not* require the messages to be explicitly signed, since the NIZK for $\mathcal{R}_{\mathsf{nom}}$ already ensures that the parties own the decryption keys corresponding to the ephemeral encryption keys.

Public verifiability. Notice that the forensic procedure IdentifyNominee does not involve any secrets of a party and could be executed by anyone. Moreover, the NIZK guarantees that $u_k \neq \perp$ if $b_k = \mathbf{true}$. Therefore, the scheme is publicly verifiable.

Acknowledgments. We thank Vincenzo Botta for valuable discussions during the early stages of this project. *Paweł Kędzior, Marcin Mielniczuk:* this work has been partially funded by the European Research Council under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666). *Daniele Venturi* is member of the Gruppo Nazionale Calcolo Scientifico Istituto Nazionale di Alta Matematica (GNCS-INdAM). His research was supported by project SERICS (PE00000014) and by project PARTHENON (B53D23013000006), under the MUR National Recovery and Resilience Plan funded by the European Union-NextGenerationEU, and by project BEAT, funded by Sapienza University of Rome.

References

ADMM14. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 443–458. IEEE Computer Society, 2014.

BCR+24. Rishabh Bhadauria, James Hsin-yu Chiang, Divya Ravi, Jure Sternad, and Sophia Yakoubov. Minimizing the use of the honest majority in YOSO MPC with guaranteed output delivery. *IACR Cryptol. ePrint Arch.*, page 2059, 2024.

BDD+20. Carsten Baum, Bernardo David, Rafael Dowsley, Ravi Kishore, Jesper Buus Nielsen, and Sabine Oechsner. Craft: Composable randomness and almost fairness from time. *IACR Cryptol. ePrint Arch.*, 2020:784, 2020.

BDD+21a. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 429–459. Springer, 2021.

BDD+21b. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 429–459. Springer, 2021.

BDF+25. Jan Bormet, Stefan Dziembowski, Sebastian Faust, Tomasz Lizurej, and Marcin Mielniczuk. Strong secret sharing with snitching. In *Annual International Cryptology Conference*, pages 295–327. Springer, 2025.

BGG+20. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*, pages 260–290, Cham, 2020. Springer International Publishing.

BGM+18. Christian Badertscher, Juan Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? a rational protocol design treatment of bitcoin. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 34–65. Springer, 2018.

BHK+22. Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. Threshold cryptography as a service (in the multiserver and YOSO models). In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 323–336. ACM, 2022.

BK14. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439. Springer, 2014.

BMSZ20. Foteini Baldimtsi, Varun Madathil, Alessandra Scafuro, and Linfeng Zhou. Anonymous lottery in the proof-of-stake setting. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 318–333, 2020.

CD24. Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 216–248. Springer, 2024.

CDGK22. Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: fast and simple encryption and secret sharing in the YOSO model. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 651–680. Springer, 2022.

CDK+22. Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 151–180. Springer, 2022.

CDPW07. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

CKR+23. Ran Canetti, Sebastian Kolby, Divya Ravi, Eduardo Soria-Vazquez, and Sophia Yakoubov. Taming adaptivity in YOSO protocols: The modular way. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part II*, volume 14370 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2023.

CM19. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

CR03. Ran Canetti and Tal Rabin. Universal composition with joint state. In *Annual International Cryptology Conference*, pages 265–281. Springer, 2003.

DFLM24. Stefan Dziembowski, Sebastian Faust, Tomasz Lizurej, and Marcin Mielniczuk. Secret sharing with snitching. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 840–853, 2024.

DKLN25. Ivan Bjerre Damgård, Simon Holmgaard Kamp, Julian Loss, and Jesper Buus Nielsen. Asynchronous YOSO a la paillier. *IACR Cryptol. ePrint Arch.*, page 128, 2025.

DP09. Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

EMP25. Daniel Escudero, Elisaweta Masserova, and Antigoni Polychroniadou. Towards scalable YOSO MPC via packed secret-sharing. *IACR Cryptol. ePrint Arch.*, page 635, 2025.

Fel23. Ed Felten. Arbitrum's fraud proofs haven't been used since it launched, September 4 2023. Accessed: 2025-09-20.

FHKS21. Sebastian Faust, Carmit Hazay, David Kretzler, and Benjamin Schlosser. Generic compiler for publicly verifiable covert multi-party computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 782–811. Springer, 2021.

GHK+21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once / secure MPC with stateless ephemeral roles. Cryptology ePrint Archive, Paper 2021/210, 2021.

GHM+21. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index pir and applications. In Kobbi Nissim and Brent Waters, editors, *TCC (3)*, volume 13044 of *Lecture Notes in Computer Science*, pages 32–61. Springer, 2021.

GOS12. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3), June 2012.
HK22. Shai Halevi and Eyal Kushilevitz. Random-index oblivious ram. In *Theory of Cryptography Conference*, pages 33–59. Springer, 2022.
JDP83. Kumar Joag-Dev and Frank Proschan. Negative association of random variables, with applications. *The Annals of Statistics*, pages 286–295, 1983.
KB14. Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 30–41. ACM, 2014.
KMTZ13. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer, 2013.
KRY24. Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Constant-round YOSO MPC without setup. *IACR Commun. Cryptol.*, 1(3):30, 2024.
LMR+24. Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Improved YOSO randomness generation with worst-case corruptions. In Jeremy Clark and Elaine Shi, editors, *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part II*, volume 14745 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2024.
LMR+25. Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Efficient distributed randomness generation from minimal assumptions where parties speak sequentially once. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part V*, volume 15605 of *Lecture Notes in Computer Science*, pages 176–206. Springer, 2025.
RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid mpc for dishonest majority. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part I*, pages 719–749, Berlin, Heidelberg, 2022. Springer-Verlag.
SSS21. Peter Scholl, Mark Simkin, and Luisa Siniscalchi. Multiparty computation with covert security and public verifiability. *Cryptology ePrint Archive*, 2021.

A Anonymous selection functionality $\mathcal{F}_{\mathsf{AS}}$

The functionality is a specialization of the functionality from [BMSZ20]. Since in our case the lottery does not depend on the stake of the parties and the nomination messages are handles separately, we removed the stakes from the functionality and the message in CreateProof. In their formalization, the *tags* are used as the common input for the parties (e.g., for a given round r with the randomness ρ, the tag is $\mathtt{tag} = (r, \rho, \dots)$.

Anonymous selection functionality $\mathcal{F}_{\mathsf{AS}}$

The ideal functionality is parameterized by an Eligible predicate and maintains the following elements:

(1) A global set of registered parties $\mathcal{P} = (P_1, \dots, P_n)$.
(2) A table T, which has one row per party and a column for each $\mathtt{tag} \in \mathbb{N}$ given by parties when checking eligibility. The table stores the eligibility information of each party in each tag.
(3) A list $\mathcal{L}$, to store a proof π corresponding to tag.

– Upon receiving (EligibilityCheck, sid, tag) from a party P_i do the following:
  1. If $P_i \in \mathcal{P}$ and $T(P_i, \mathtt{tag})$ is undefined, sample $r \in \{0,1\}^\ell$, run Eligible$(r, \mathtt{tag})$ to get $b \in \{0,1\}$. Set $T(P_i, \mathtt{tag}) \leftarrow b$.
  2. Output (EligibilityCheck, $T(P_i, \mathtt{tag})$) to P_i.

– Upon receiving (CreateProof, sid, tag) from P_i:
  1. If $T(P_i, \mathtt{tag}) = 1$, send (Prove, tag) to $\mathcal{A}$. Else, send (Declined, tag) to P_i.
  2. Upon receiving (Done, ψ, tag) from $\mathcal{A}$, set $\pi \leftarrow \psi$ and record $(\pi, \mathtt{tag})$ in $\mathcal{L}$. Send (Proof, π, tag) to P_i.

– Upon receiving (VERIFY, sid, π, tag) from some party P':
  1. If $(\pi, \mathtt{tag}) \in \mathcal{L}$ output (VERIFIED, sid, $(\pi, \mathtt{tag}), 1$) to P'.
  2. If $(\pi, \mathtt{tag}) \notin \mathcal{L}$, send (VERIFY, sid, $(\pi, \mathtt{tag})$) to $\mathcal{A}$ and wait for a witness w from the adversary $\mathcal{A}$. Check if w is valid as follows:
     – Parse $w = (P_i, \mathtt{tag})$ and check that $T(P_i, \mathtt{tag}) = 1$.
     – If yes, store $(\pi, \mathtt{tag})$ in the list $\mathcal{L}$ and send (VERIFIED, sid, $(\pi, \mathtt{tag}), 1$) to P'.

     If either of these checks are false output (VERIFIED, sid, $(\pi, \mathtt{tag}), 0$) to P'.

B The time-lock puzzle functionality $\mathcal{F}_{\mathsf{TLP}}$

The functionality is taken verbatim from [BDD+21b].

Anonymous selection functionality $\mathcal{F}_{\mathsf{TLP}}$

$\mathcal{F}_{\mathsf{TLP}}$ is parameterized by a set of parties $\mathcal{P}$, an owner $\mathcal{P}_o \in \mathcal{P}$, a computational security parameter τ, a state space $\mathcal{ST}$ and a tag space $\mathcal{TAG}$.
In addition to $\mathcal{P}$, the functionality interacts with an adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{TLP}}$ contains initially empty lists **steps** (honest puzzle states), **omsg** (output messages), **in** (inbox) and **out** (outbox).

Create puzzle: Upon receiving the first message (CREATEPUZZLE, sid, Γ, m) from $\mathcal{P}_o$ where $\Gamma \in \mathbb{N}^+$ and $m \in \{0,1\}^\tau$, proceed as follows:

1. If $\mathcal{P}_o$ is honest, sample $\mathtt{tag} \xleftarrow{\$} \mathcal{TAG}$ and $\Gamma+1$ random distinct states $\mathsf{st}_j \xleftarrow{\$} \{0,1\}^\tau$ for $j \in \{0, \ldots, \Gamma\}$. If $\mathcal{P}_o$ is corrupted, let $\mathcal{S}$ provide values $\mathtt{tag} \in \mathcal{TAG}$ and $\Gamma+1$ distinct values $\mathsf{st}_j \in \mathcal{ST}$.
2. Append $(\mathsf{st}_0, \mathtt{tag}, \mathsf{st}_\Gamma, m)$ to **omsg**, append $(\mathsf{st}_j, \mathsf{st}_{j+1})$ to **steps** for $j \in \{0, \ldots, \Gamma - 1\}$, and output (CREATEDPUZZLE, sid, puz $= (\mathsf{st}_0, \Gamma, \mathtt{tag}),$) to $\mathcal{P}_o$ and $\mathcal{S}$. $\mathcal{F}_{\mathsf{TLP}}$ stops accepting messages of this form.

Solve: Upon receiving (SOLVE, sid, st) from party $\mathcal{P}_i \in \mathcal{P}$ with $\mathsf{st} \in \mathcal{ST}$, if there exists $(\mathsf{st}, \mathsf{st}') \in \mathtt{steps}$, append $(\mathcal{P}_i, \mathsf{st}, \mathsf{st}')$ to **in** and ignore the next steps. If there is no $(\mathsf{st}, \mathsf{st}') \in \mathtt{steps}$, proceed as follows:

– If $\mathcal{P}_o$ is honest, sample $\mathsf{st}' \xleftarrow{\$} \mathcal{ST}$.
– If $\mathcal{P}_o$ is corrupted, send (SOLVE, sid, st) to $\mathcal{S}$ and wait for answer (SOLVE,sid, st, st′).

Append $(\mathsf{st}, \mathsf{st}')$ to **steps** and append $(\mathcal{P}_i, \mathsf{st}, \mathsf{st}')$ to **in**.

Get Message: Upon receiving (GETMSG, sid, puz, st) from party $\mathcal{P}_i \in \mathcal{P}$ with $\mathsf{st} \in \mathcal{ST}$, parse $\mathsf{puz} = (\mathsf{st}_0, \Gamma, \mathtt{tag})$ and proceed as follows:

– If $\mathcal{P}_o$ is honest and there is no $(\mathsf{st}_0, \mathtt{tag}, \mathsf{st}, m) \in \mathtt{omsg}$, append $(\mathsf{st}_0, \mathtt{tag}, \mathsf{st}, \bot)$ to omsg.
– If $\mathcal{P}_o$ is corrupted and there exists no $(\mathsf{st}_0, \mathtt{tag}, \mathsf{st}, m) \in \mathtt{omsg}$, send (GETMSG,sid, puz, st) to $\mathcal{S}$, wait for $\mathcal{S}$ to answer with (GETMSG,sid, puz, st, m) and append $(\mathsf{st}_0, \mathtt{tag}, \mathsf{st}, m)$ to omsg.

Get $(\mathsf{st}_0, \mathtt{tag}, \mathsf{st}, m)$ from omsgand output (GETMSG, sid, $\mathsf{st}_0, \mathtt{tag}, \mathsf{st}, m$) to $\mathcal{P}_i$.

Output: Upon receiving (OUTPUT, sid) by $\mathcal{P}_i \in \mathcal{P}$, retrieve the set L_i of all entries $(\mathcal{P}_i, \cdot, \cdot)$ in **out**, remove L_i from **out** and output (COMPLETEsid, L_i) to $\mathcal{P}_i$.

Tick: Set **out** $\leftarrow$ **in** and set **in** $= \emptyset$.

C The functionality $\mathcal{F}_{\mathsf{NIZK}}$

The functionality is taken verbatim from [GOS12].

Non-interactive Zero-knowledge Proofs functionality $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}]$.

$\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}]$ is parameterized by polynomial-time decidable relation $\mathcal{R} \in \{0,1\}^* \times \{0,1\}^*$, and runs with parties $P_1, \ldots, P_N$, and ideal process adversary $\mathcal{S}$. It stores initially empty proof table $\mathcal{Q}$.

- **Proof** Upon receiving input $(\textsc{Prove}, \mathsf{sid}, \mathsf{ssid}, x, w)$ from a party P_i, ignore if $(x, w) \notin \mathcal{R}$. Otherwise, send $(\textsc{Prove}, \mathsf{sid}, x)$ to $\mathcal{S}$. Upon receiving $(\textsc{Proof}, \pi)$ from $\mathcal{S}$, store (x, π) in $\mathcal{Q}$ and send $(\textsc{Proof}, \mathsf{sid}, \mathsf{ssid}, \pi)$ to P_i.
- **Verification** Upon receiving input $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, x, \pi)$ from a party P_i if (x, π) is not stored in $\mathcal{Q}$, then send $(\textsc{Verify}, \mathsf{sid}, x, \pi)$ to $\mathcal{S}$. Upon receiving $(\textsc{Witness}, w)$ from $\mathcal{S}$, if $(x, w) \in \mathcal{R}$, store $(x, \pi) \in \mathcal{Q}$. Finally, return $(\textsc{Verification}, \mathsf{sid}, \mathsf{ssid}, (x, \pi) \in \mathcal{Q})$ to P_i.

D The bulletin board functionality $\mathcal{F}_{\mathsf{BB}}$

The functionality is taken verbatim from [CKR+23].

Bulletin Board functionality $\mathcal{F}_{\mathsf{BB}}$

The functionality admits a finite message space Msg and we let $\mathsf{Msg}_\perp = \mathsf{Msg} \cup \{\perp\}$.

- **Initially:** Create broadcast maps:
  $m \colon \mathbb{N} \times \text{Machine} \to \mathsf{Msg}_\perp$ where $m(r, M) = \perp$ for all r, M.
- **On input** $(\textsc{Write}, \mathsf{sid}, msg)$ from machine M in round r:
  - Update $m(r, M) := msg$.
  - Store the broadcast message from the role.
  - Output $\textsc{Write}(\mathsf{sid}, m)$ to $\mathcal{S}$.
- **On input** $(\textsc{Read}, \mathsf{sid}, r')$ from machine M in round r where $r' < r$, output the set of all elements (M', r', msg) where $msg = m(r', M') \neq \perp$ to M.

E The complete functionality $\mathcal{F}_{\mathsf{ERA}}$

Building upon the functionality $\mathcal{F}_{\mathsf{RA}}$ proposed in [CKR+23], we introduce additional components (marked in gray) , thereby enabling identifiable abort in $\mathcal{F}_{\mathsf{ERA}}$. The remaining parts of the functionality are taken verbatim from [CKR+23].

Functionality $\mathcal{F}_{\mathsf{ERA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{D}, \mathsf{delay}, \delta)$

This functionality is synchronous. Following the synchronous model of Katz *et al.* [KMTZ13], and proceeding in lock-step with the protocols that use it. It has the following parameters:

- $\mathcal{P}$: the set of machines.
- n: the (maximal) size of a committee.
- $\mathcal{T}$: the function determining the number of allowed corruptions in a committee (based on the fraction of corrupt machines).
- $\mathcal{D}$: denoting a sampling algorithm, and
- delay: denoting the upper bound on the number of rounds required to complete nomination.
- $\delta \geq \mathtt{delay}$: the minimum number of rounds for which the identities of nominees remain hidden.

Init: The functionality maintains a partition of $\mathcal{P}$ into sets $\mathcal{H}$ and Cor of all honest and corrupt party identifiers, respectively. It also maintains a global set Ready initially equal to $\mathcal{P}$.

Newcommittee: After receiving $(\textsc{New}, \mathsf{cid}, C)$ from all honest parties in round r, store $(r, \mathsf{cid}, C, \mathsf{PKeys} = \emptyset, \mathsf{SKeys} = \emptyset, \mathsf{cor} = 0, \mathsf{nom} = 0, fin = \perp)$. Ignore the command if any value is already stored for cid.

- The lists PKeys and SKeys are initially empty. The list PKeys would be updated with tuples (ek, vk, R) where (ek, vk) refer to the public keys established for a role R. The list SKeys would be updated with tuples $(\mathsf{pid}, dk, sk, \mathsf{R})$ where (dk, sk) refer to the secret keys corresponding to the role R, which has been assigned to machine with identifier pid.
- The corruption and nomination counters, cor and nom, start at zero.
- A committee-specific ready set $\mathsf{Ready}_{\mathsf{cid}}$ is initialized the same as the global ready set: $\mathsf{Ready}_{\mathsf{cid}} = \mathsf{Ready}$.
- Finally, the flag signaling whether nomination is completed or not is initially false: $fin = \bot$.

Each time an honest party inputs $(\textsc{New}, \mathsf{cid}, C)$, forward this to $\mathcal{S}$.
The functionality proceeds to the next round if $fin = \top$ for all stored committees with round number $r' \leq r - \mathsf{delay}$.

Read: On input $(\textsc{Read}, \mathsf{cid}, r)$ from M with identifier pid in round r, retrieve the value $(r^*, \mathsf{cid}^*, C, \mathsf{PKeys}, \mathsf{SKeys}, \mathsf{cor}, \mathsf{nom}$ where $\mathsf{cid} = \mathsf{cid}^*, r^* \leq r - \mathsf{delay}$ and $fin = \top$. If no such value exists, do nothing.

- Collect all values $(\mathsf{pid}^*, dk, sk, \mathsf{R})$ in SKeys where $\mathsf{pid}^* = \mathsf{pid}$ into a list SKeys'. If these exist, remove pid from Ready.
- Output $(\mathsf{PKeys}, \mathsf{SKeys}')$ to M.

Delete: On input $(\textsc{Delete})$ from M with identifier pid, do the following:

- Overwrite all elements of SKeys of the form $(\mathsf{pid}^*, dk, sk, \mathsf{R})$, where $\mathsf{pid}^* = \mathsf{pid}$, with $(\mathsf{pid}^*, \bot, \bot, \mathsf{R})$.
- Set $\mathsf{Ready} \leftarrow \mathsf{Ready} \setminus \{\mathsf{pid}\}$.
- Set $\mathsf{Ready}_{\mathsf{cid}} \leftarrow \mathsf{Ready}_{\mathsf{cid}} \setminus \{\mathsf{pid}\}$ for cid with $fin = \bot$.

Ready: On input $(\textsc{Ready})$ from M with identifier pid, update the global ready set $\mathsf{Ready} \leftarrow \mathsf{Ready} \cup \{\mathsf{pid}\}$ in the beginning of the subsequent round. Output $(\textsc{Ready}, \mathsf{pid})$ to $\mathcal{S}$.

The simulator must perform nominations for each committee, but is restricted by the number of nominations it may bias relative to the current fraction of corrupt machines.

Nominate honest: On input $(\textsc{Nom-Honest}, \mathsf{cid})$ from $\mathcal{S}$ in round r, retrieve the value $(r, \mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys}, \mathsf{cor}, \mathsf{nom},$ If no such value exists do nothing. If $\mathsf{nom} < c$, do the following:

- Update $\mathsf{nom} \leftarrow \mathsf{nom} + 1$.
- Generate fresh encryption and signing keys for the chosen machine: $(ek, dk) \leftarrow \mathsf{PKE.Gen}()$, $(vk, sk) \leftarrow \mathsf{SIG.Gen}()$.
- Append $(ek, vk, \bot)$ to PKeys.
- Add $(\bot, dk, sk, \bot)$ to SKeys.
- If $\mathsf{nom} = c$, go to procedure $\textsc{Finish}(\mathsf{cid})$.
- Output $(\textsc{Nom-Honest}, \mathsf{cid}, ek, vk)$ to $\mathcal{S}$.

Nominate corrupt: On input $(\textsc{Nom-Corrupt}, \mathsf{cid}, \mathsf{pid}, (ek, vk), (dk, sk))$ from $\mathcal{S}$ in round r, retrieve the value $(r, \mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys}, \mathsf{cor}, \mathsf{nom}, fin)$. If no such value exists, do nothing. If $\mathsf{nom} < n$ and $\mathsf{cor} + 1 < \mathcal{T}(|\mathsf{Cor}|/|\mathcal{P}|)$, do the following:

- Update the nominated and corrupt counters $\mathsf{nom} \leftarrow \mathsf{nom} + 1$, $\mathsf{cor} \leftarrow \mathsf{cor} + 1$.
- Verify that $(ek, dk) \neq (\bot, \bot)$ and $(vk, sk) \neq (\bot, \bot)$ indeed form key pairs. If any of the checks fails, return.
- Append $(ek, vk, \bot)$ to PKeys and $(\mathsf{pid}, dk, sk, \bot)$ to SKeys.
- If $\mathsf{nom} = n$, go to procedure $\textsc{Finish}(\mathsf{cid})$.

Corrupt nominee: On input (CORRUPT-NOMINEE, cid, pid) from $\mathcal{S}$ in round r, retrieve the value $(r', \mathsf{cid}', C, \mathsf{PKeys}, \mathsf{SKeys}, \mathsf{cor}, \mathsf{nom}, fin)$ where $\mathsf{cid} = \mathsf{cid}'$ and $r = r'$. If no such value exists, do nothing. If $\mathsf{cor} + 1 < \mathcal{T}(|\mathsf{Cor}|/|\mathcal{P}|)$ and $\mathsf{cor} < \mathsf{nom}$, do the following:

- $\mathsf{cor} \leftarrow \mathsf{cor} + 1$
- Choose an element $(\mathsf{pid}', dk, sk, \perp)$ uniformly at random between the values of SKeys where $\mathsf{pid}' = \perp$
- Update this value to be $(\mathsf{pid}, dk, sk, \perp)$
- Output (CORRUPT-NOMINEE, cid, pid, dk, sk) to $\mathcal{S}$

Finish (cid): When the procedure FINISH(cid) is called, retrieve the value $(r', \mathsf{cid}', C, \mathsf{PKeys}, \mathsf{SKeys}, \mathsf{cor}, \mathsf{nom}, fin)$ where $\mathsf{cid}' = \mathsf{cid}$ and do the following:

- Sample a random permutation ϕ on $[c]$
- For the i-th element of PKeys update $(ek, vk, \perp)$ to $(ek, vk, C_{\phi(i)})$
- For the i-th element of SKeys update $(\mathsf{pid}, dk, sk, \perp)$ as follows:
  - If $\mathsf{pid} = \perp$, choose an honest machine uniformly at random as $\mathsf{pid}' \leftarrow \mathcal{D}(\mathcal{H}, \mathcal{P})$. If $\mathsf{pid}' \in \mathsf{Ready}_{\mathsf{cid}}$, update to $(\mathsf{pid}', dk, sk, C_{\phi(i)})$
  - Else, update to $(\mathsf{pid}, dk, sk, C_{\phi(i)})$
- Set $fin = \top$ for cid
- Output (FINISH, cid, ϕ, PKeys) to $\mathcal{S}$ when finished

Corrupt: Upon receiving (CORRUPT, pid) from $\mathcal{E}$, output all elements $(\mathsf{pid}', dk, sk, \mathsf{R})$ of any stored SKeys, where $\mathsf{pid}' = \mathsf{pid}$ to $\mathcal{S}$.

Identify nominee: On input (IDENTIFYNOMINEE, cid, r, ek) from M in round $r' \geq r + 1$, send (IDENTIFYNOMINEE, cid, r, ek) to $\mathcal{S}$ and wait until round $r' + \delta$ and continue accepting commands in parallel.

In round $r' + \delta$, send (IDENTIFYNOMINEE, cid, r, ek, pid) to $\mathcal{S}$ and M, where pid is the unique value such that $(\mathsf{ek}, \mathsf{vk}, R) \in \mathsf{PKeys}$ and $(\mathsf{pid}, _, _, R) \in \mathsf{SKeys}$, with $\mathsf{ek}, \mathsf{vk} \neq \perp$.

F The complete protocol Π_{ERA}

In this section we turn to the construction of a protocol Π_{ERA} that realizes the extended functionality, where all modifications to the original protocol Π_{RA} introduced in [CKR+23] are highlighted in gray , whereas the remaining parts of the protocol are taken verbatim from [CKR+23].

Protocol Π_{ERA} in the $(\mathcal{F}_{\mathsf{Beacon}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{AS}}, \mathcal{F}_{\mathsf{TLP}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid world

Each machine M has access to a PKI containing KM-NCE public keys for each computation machine. Each role stores its current long-term KM-NCE secret key as $M.sk$.Let n be the predefined size of a committee. We instantiate $\mathcal{F}_{\mathsf{AS}}$ with selection probability s (e.g., $s = (1 + \eta)c/|\mathcal{P}|$), i.e., in the ELIGIBILITYCHECK procedure $\mathcal{F}_{\mathsf{AS}}$ samples $r \leftarrow \{0,1\}^\ell$ and sets $\mathsf{Eligible}(r, \mathsf{tag}) = 1$ iff $r < s \cdot 2^\ell$.

All other ideal functionalities are used with their default parameterizations.

New Committee: After receiving input (NEW, cid, C) in round r, machine M with identifier pid performs the following procedure:

- If there already exists stored value with $\mathsf{cid}^* = \mathsf{cid}$ ignore this command. Otherwise, store the value $(r, \mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys})$, where PKeys and SKeys are empty lists.
- Input (READ, r) to the randomness beacon, to receive randomness ρ.
- Send (ELIGIBILITYCHECK, cid, $\mathsf{tag} = (r, \mathsf{pid}, \mathsf{cid}, \rho)$) to $\mathcal{F}_{\mathsf{AS}}$ to obtain draw.
- If draw is a winning draw (i.e., $\mathsf{draw} = 1$), send (CREATEPROOF, cid, tag) to $\mathcal{F}_{\mathsf{AS}}$ and obtain the proof π and proceed to nominate a party, otherwise skip the remaining steps

- Sample a uniformly random machine index $P_j = \mathsf{pid} \leftarrow\!\!\$\ \mathcal{P}$.
- Generate fresh ephemeral encryption and signing keys for the nominated role, $(ek, dk) \leftarrow \mathsf{PKE.Gen}()$ $(vk, sk) \leftarrow \mathsf{SIG.Gen}()$.
- Encrypt the decryption and signing key to the chosen machine $\mathsf{ct} \leftarrow \mathsf{KM{-}NCE.Enc}(M.pk, (\mathsf{pid}, dk, sk))$.
- Send $(\textsc{CreatePuzzle}, \mathsf{cid}, \delta, \mathsf{pid})$ to $\mathcal{F}_{\mathsf{TLP}}$.
- On $(\textsc{CreatedPuzzle}, \mathsf{cid}, \mathsf{tlp})$ from $\mathcal{F}_{\mathsf{TLP}}$, let $x = (\mathsf{ek}, \mathsf{vk}, \mathsf{setP}, \mathsf{ct}, \mathsf{tlp}, \delta)$ and $w = (\mathsf{dk}, \mathsf{sk}, j, P_j.\mathsf{pk}, r_{\mathsf{ct}})$, where r_{ct} is the randomness used to create ct. Send $(\textsc{Prove}, \mathsf{cid}, \mathsf{ssid}, x, w)$ to $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$.
- On $(\textsc{Proof}, \mathsf{cid}, \mathsf{ssid}, \pi^{\mathsf{nom}})$ erase the keys dk, sk and all randomness used for sampling the keys and pid, as well as any encryption randomness.
- Post $(\mathsf{cid}, \mathsf{ek}, \mathsf{vk}, \mathsf{ct}, \mathsf{draw}, \pi, \mathsf{tlp}, \pi^{\mathsf{nom}})$ to the bulletin board.

Read: On input $(\textsc{Read}, \mathsf{cid}, r')$ in round r' where $r + 2 \leq r'$:

1. Retrieve the value $(r, \mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys})$, stopping if no such value exists.
2. Observe the bulletin board and collect a list of messages for committee identifier cid posted in round r, that is, let $L = (l_1, \ldots, l_k)$ where $l_j = (\mathsf{cid}, \mathsf{ek}_j, \mathsf{vk}_j, \mathsf{ct}_j, \mathsf{draw}_j, \pi_j, \mathsf{tlp}_j, \pi_j^{\mathsf{nom}})$
3. Input $(\textsc{Read}, r)$ to the randomness beacon, to receive randomness ρ.
4. Remove from L any elements $l_j = (\mathsf{cid}, \mathsf{ek}_j, \mathsf{vk}_j, \mathsf{ct}_j, \mathsf{draw}_j, \pi_j)$ posted by machine M from the list where draw_j is not a winning draw, that is, $\mathcal{F}_{\mathsf{AS}}$ on $(\textsc{Verify}, \mathsf{cid}, \mathsf{tag} = (r, \mathsf{pid}, \mathsf{cid}, \rho), \pi_j)$ replies with $(\textsc{Verify}, \mathsf{cid}, \mathsf{tag} = (r, \mathsf{pid}, \mathsf{cid}, \rho), \pi_j, 0)$.
   For the remaining elements, let $x_j = (\mathsf{ek}, \mathsf{vk}, \mathsf{setP}, \mathsf{ct}_j, \mathsf{tlp}_j, \delta)$ and send $(\textsc{Verify}, \mathsf{cid}, \mathsf{ssid}, x_j, \pi_j^{\mathsf{nom}})$ to $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$. Remove the element from the list if $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$ rejects, i.e., sends $(\textsc{Verification}, \mathsf{cid}, \mathsf{ssid}, \mathbf{false})$.
5. ~~Sort the list lexicographically by encryption key, keeping only the n first elements. If the list does not have exactly n elements pad it with values $(\mathsf{cid}, \bot, \bot, \bot)$.~~
6. Input $(\textsc{Read}, r + 1)$ to the randomness beacon, to receive randomness ρ'.
7. Let σ be a uniformly random permutation on $[|L|]$ defined by the randomness ρ' and apply σ to the list.
8. Keep only the first n elements of the permuted list. If the list does not have exactly n elements, abort.
9. Loop over the list, for the j-th element $(\mathsf{cid}, \mathsf{ek}_j, \mathsf{vk}_j, \mathsf{ct}_j, \ldots)$:
   - Append $(\mathsf{ek}_j, \mathsf{vk}_j, C_j)$ to PKeys.
   - Attempt to decrypt $(\mathsf{pid}, \mathsf{dk}, \mathsf{sk}) \leftarrow \mathsf{KM\text{-}NCE.Dec}(M_j, \mathsf{sk}, \mathsf{ct}_j)$. If $(\mathsf{pid}, \mathsf{dk}, \mathsf{sk}) \neq \bot$ and pid matches the machine which posted the element to the bulletin board, append $(\mathsf{pid}, \mathsf{dk}, \mathsf{sk}, C_j)$.
10. Output PKeys and SKeys to M.

Delete: When given input $\textsc{Delete}$, for each stored value $(r, \mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys})$ delete SKeys overwriting it with the empty list. Finally, delete the long term secret key $M.\mathsf{sk}$.
Ready: When given input $\textsc{Ready}$, generate a new key pair $(\mathsf{pk}, \mathsf{sk}, tk) \leftarrow \mathsf{KM\text{-}NCE.Gen}()$, setting $M.\mathsf{sk} = \mathsf{sk}$ and deleting tk immediately. Finally, post $(\mathsf{pid}, \mathsf{pk})$ to the bulletin board.

Identify Nominee: Party P_i on input $(\textsc{IdentifyNominee}, \mathsf{cid}, r, \mathsf{ek})$ from the environment $\mathcal{Z}$ in round $r' \geq r+1$, send $\textsc{Read}(\mathsf{bid}, r)$ to $\mathcal{F}_{\mathsf{BB}}$ and retrieve an entry of the form $(\mathsf{bid}, r, \mathsf{ek}, \mathsf{vk}, c, \mathsf{draw}, \pi, \mathsf{tlp}, \pi^{\mathsf{nom}})$. If no such entry exists, return $\perp$ immediately. Parse $\mathsf{tlp} =: (\mathsf{st}_0, \Gamma, \mathtt{tag})$.

1. Set $\mathsf{st} := \mathsf{st}_0$.
2. **For** $t = 1$ **to** δ **do**
   (a) Send $(\textsc{Solve}, \mathsf{sid}, \mathsf{st})$ to $\mathcal{F}_{\mathsf{TLP}}$.
   (b) Wait until the next round, i.e., round $r' + t$.
   (c) Send $(\textsc{Output}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{TLP}}$ and receive $(\textsc{Complete}, \mathsf{sid}, L_i)$.
   (d) If there exists $(\mathsf{P}_i, \mathsf{sid}, \mathsf{st}, \mathsf{st}') \in L_i$, then set $\mathsf{st} := \mathsf{st}'$.
3. Send $(\textsc{GetMsg}, \mathsf{sid}, \mathsf{st}_0, \mathsf{tlp}, \mathsf{st})$ to $\mathcal{F}_{\mathsf{TLP}}$.
4. On $(\textsc{GetMsg}, \mathsf{sid}, \mathsf{st}_0, tag, \mathsf{st}, \mathsf{pid})$ with $\mathsf{pid} \neq \perp$ and **output** pid to everyone in round $r' + \delta$.

G The complete simulator $\mathcal{S}_{\mathsf{ERA}}$

In this section we turn to the construction of a simulator $\mathcal{S}_{\mathsf{ERA}}$ for the extended functionality, where all modifications to the original simulator $\mathcal{S}_{\mathsf{ERA}}$ introduced in [CKR+23] are highlighted in gray, whereas the remaining parts of the simulator are taken verbatim from [CKR+23].

Simulator $\mathcal{S}_{\mathsf{ERA}}$

The simulator interacts with the ideal functionality $\mathcal{F}_{\mathsf{ERA}}$.
We work with a TLP idealization that provides the simulator with a concealed *setup trapdoor* enabling equivocation of puzzles. Using this trapdoor, the simulator generates puzzle headers that are indistinguishable from honest ones but are initially bound to a dummy payload pid'. At the designated release time (in round δ), once the ideal functionality provides the true message pid, the simulator applies the trapdoor operation to *rebind* the existing puzzle so that any legitimate solve yields pid (and not pid'). Since this trapdoor exists only in the ideal setup and is unavailable in real executions, the simulated transcript remains consistent with the protocol while preserving indistinguishability from the adversary's point of view. In addition, the simulator uses the TLP extraction trapdoor for π^{nom} to extract a nomination witness from each corrupt nomination and provide it to the ideal functionality. This guarantees that committees can always be completed (by re-nomination when necessary) and that any aborting nominee is accountable and can be punished. The simulator also maintains a list L containing the identifiers of pre-committee members, initialized as empty. We omit the details of the simulation of trivial communication between the simulator and auxiliary functionalities (e.g., $\mathcal{F}_{\mathsf{AS}}$), as they are straightforward and do not involve any non-trivial programming or extraction.

1. For each party $p \in \mathcal{H}$, the simulator generates long term public and private keys $(\mathsf{ltsk}_p, \mathsf{ltpk}_p)$, send public keys to the bulletin board and records the public keys of corrupted parties.
2. When an honest machine with identifier pid given $(\textsc{New}, \mathsf{cid}, C)$ wins the sortition (by interaction with F_{AS}), the simulator $\mathcal{S}_{\mathsf{ERA}}$ does the following:
   - Add pid to the list L.
     Choose a dummy value $\mathsf{m_D}$ and send $(\textsc{CreatePuzzle}, (\mathsf{cid}, j), \delta, \mathsf{m_D})$, where j is number of realized honest nominations. On $(\textsc{CreatedPuzzle}, (\mathsf{cid}, j), \mathsf{tlp})$ from $\mathcal{F}_{\mathsf{TLP}}$, let $x^* = (\mathsf{ek}^*, \mathsf{vk}^*, \mathsf{setP}, \mathsf{ct}^*, \mathsf{tlp}, \delta)$ and $w = (\mathsf{dk}^*, \mathsf{sk}^*, j^*, P_j^*.\mathsf{pk}, r_{\mathsf{ct}}^*)$, where all this values are dummy values. Send $(\textsc{Prove}, \mathsf{cid}, \mathsf{ssid}, x, w)$ to $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$.
   - On $(\textsc{Proof}, \mathsf{cid}, \mathsf{ssid}, \pi^{\mathsf{nom}})$ from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$ save $(\mathsf{cid}, j, \mathsf{tlp}, \pi^{\mathsf{nom}})$
     Upon receiving $(\textsc{Nom-Honest}, \mathsf{cid}, \mathsf{ek}, \mathsf{vk})$ retrieve $(\mathsf{cid}, j, \pi^{\mathsf{nom}})$ and post $(\mathsf{cid}, \mathsf{ek}, \mathsf{vk}, \mathsf{ct}, \mathsf{draw}, \pi, \mathsf{tlp}, \pi^{\mathsf{nom}})$ to the bulletin board, where ct is a faked ciphertext.
   - Otherwise, choose a uniformly random corrupt party $\mathsf{pid} \leftarrow \mathsf{Cor}$, and sample fresh ephemeral encryption and signing keys for the nominated role, $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.Gen}()$ $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Gen}()$ and

send $(\text{NOM-CORRUPT}, \mathsf{cid}, \mathsf{pid}, (\mathsf{ek}, \mathsf{vk}), (\mathsf{dk}, \mathsf{sk}))$ to $\mathcal{F}_{\mathsf{ERA}}$. Post $(\mathsf{cid}, \mathsf{ek}, \mathsf{vk}, ct, \mathsf{draw}, \pi, \mathsf{tlp}, \pi^{\mathsf{nom}})$ to the bulletin board, where ct is a real encryption of the keys.

3. When $\mathcal{A}_{\mathsf{ERA}}$ posts ciphertext $(\mathsf{cid}, \mathsf{ek}, \mathsf{vk}, ct, \mathsf{draw}, \pi, \mathsf{tlp}, \pi^{\mathsf{nom}})$ to the bulletin board, the simulator attempts to decrypt ct for each honest machine M_j. If $(\mathsf{pid}^*, \mathsf{dk}, \mathsf{sk}) = \mathsf{KM-NCE.Dec}(M_j.\mathsf{ltsk}, ct)$, where pid^* is the identifier for the corrupt machine which posted the message, then input Add pid^* to the list L. Abort, if the ciphertext correctly decrypts for more than one honest party. If it only decrypts to $\perp$ for honest parties, Run the extractor of $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$ on π^{nom} to obtain a witness w_{ext}. Parse w_{ext} as $(\mathsf{dk}_{\text{ext}}, \mathsf{sk}_{\text{ext}}, j, \mathsf{pid}_{\text{ext}}, r_{\mathsf{ct}})$. If extraction fails or the witness is invalid, abort the simulation. Add $\mathsf{pid}_{\text{ext}}$ to the list L and store $(\mathsf{dk}_{\text{ext}}, \mathsf{sk}_{\text{ext}})$ for future use in step 6.
4. When $\mathcal{A}_{\mathsf{ERA}}$ performs an adaptive corruption on the machine with identifier pid, for each honest nomination in a committee assign the nomination to the machine with probability $\frac{1}{|\mathcal{H}|}$. When a role is assigned to the machine input $(\text{CORRUPT-NOMINEE}, \mathsf{cid}, \mathsf{pid})$ to the functionality to receive secret keys for the role in $(\text{CORRUPT-NOMINEE}, \mathsf{cid}, \mathsf{pid}, \mathsf{dk}, \mathsf{sk})$. Use the open algorithm to generate the KM-NCE secret keys such that the machine can decrypt the corresponding ciphertext. If the machine is assigned more than k roles the simulator will be unable to open the faked ciphertexts and must abort.
5. When proceeding to the next round abort if the committee has fewer than n roles.
   When proceeding to the next round, the simulator chooses a random sublist L' of L of length n. If L has fewer than n elements, abort the simulation.
   For each honest pid in L', send $(\text{NOM-HONEST}, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{ERA}}$. For each corrupt pid in L', send $(\text{NOM-CORRUPT}, \mathsf{cid}, \mathsf{pid}, (\mathsf{ek}, \mathsf{vk}), (\mathsf{dk}, \mathsf{sk}))$ to $\mathcal{F}_{\mathsf{ERA}}$, where dk, sk were stored in step 3 for the corresponding pid. Since $|L'| = c$, this causes the functionality to finish, outputting $(\text{FINISH}, \mathsf{cid}, \phi, \mathsf{PKeys})$, where ϕ fixes the permutation between keys and roles. The simulator should then reprogram the
6. randomness beacon for round $r + 1$ to match ϕ and the simulated choice of L'.
7. On $(\text{IDENTIFYNOMINEE}, \mathsf{cid}, r, \mathsf{ek})$ from $\mathcal{F}_{\mathsf{ERA}}$ retrieve $(\mathsf{cid}, j, \mathsf{tlp}, \pi^{\mathsf{nom}})$ related to ek. Parse $\mathsf{tlp} =: (\mathsf{st}_0, \delta, \mathsf{tag})$

   (a) Set $\mathsf{st} := \mathsf{st}_0$.
   (b) **For** $t = 1$ **to** δ **do**
      i. Send $(\text{SOLVE}, (\mathsf{cid}, i), \mathsf{st})$ to $\mathcal{F}_{\mathsf{TLP}}$.
      ii. Wait until round $r' + t$.
      iii. Send $(\text{OUTPUT}, (\mathsf{cid}, i))$ to $\mathcal{F}_{\mathsf{TLP}}$ and receive $(\text{COMPLETE}, (\mathsf{cid}, i), L_i)$.
      iv. If there exists $(\mathsf{P}_i, \mathsf{sid}, \mathsf{st}, \mathsf{st}') \in L_i$, then set $\mathsf{st} := \mathsf{st}'$.
   On $(\text{IDENTIFYNOMINEE}, \mathsf{cid}, r, \mathsf{ek}, \mathsf{pid})$ program $\mathcal{F}_{\mathsf{TLP}}$ to output $(\text{GETMSG}, (\mathsf{cid}, i), \mathsf{tlp}, \mathsf{st}, \mathsf{pid})$

The simulator fails if it is forced to make too many corrupt nominations and therefore aborts if more than $\mathcal{T}(f)$ corruptions are made in any committee.

H The complete security proof of Π_{ERA}

In this section we present proof of Theorem 2. All modifications to the original design proof presented in [CKR+23] are highlighted in gray and the rest are taken verbatim from [CKR+23]. In addition to the hybrids from [CKR+23], our proof introduces two new hybrids, H_2 and H_3, which address the use of time-lock puzzles. In H_2, we replace honest encryptions by dummy locks on $\mathsf{m}_{\mathsf{D}} := 0^{|m|}$ while recording the real message in $\mathcal{F}_{\mathsf{TLP}}$, and in H_3 we eliminate the concrete TLP entirely, replacing it with synthetic puzzle headers and direct interaction with $\mathcal{F}_{\mathsf{TLP}}$. These steps are required to capture the TLP functionality explicitly, while the remainder of the hybrid argument follows directly from the framework of [CKR+23].

Hybrids. We now define the hybrid experiments used in the proof.

H_0: Run the real experiment $\mathsf{REAL}_{\Pi_{\mathsf{nat}}, A_{\mathsf{nat}}, E_{\mathsf{nat}}}(1^\kappa)$.

H_1: For honest parties $\mathcal{S}_{\mathsf{ERA}}$ replaces tlp with dummy tlp^*, by sending $(\textsc{CreatePuzzle}, (\mathsf{cid}, j), \delta, \mathsf{m_D})$ to $\mathcal{F}_{\mathsf{TLP}}$. After δ rounds when $\mathcal{S}_{\mathsf{ERA}}$ gets correct pid from $\mathcal{F}_{\mathsf{ERA}}$, respond with $(\textsc{GetMsg}, (\mathsf{cid}, i), \mathsf{tlp}^*, \mathsf{st}, \mathsf{pid})$.

H_2: We eliminate concrete TLP entirely: honest parties send puzzles $\mathsf{puz}^* = (\mathsf{st}_0^*, \delta, \mathsf{tag}^*)$ where st_0^* and tag^* are values drawn from $\{0,1\}^\tau$ and $\mathcal{TAG}$ respectively, and interact only with $\mathcal{F}_{\mathsf{TLP}}$ for storing and later releasing pid.

H_3: ~~The same as previous, except the simulator aborts if the corrupt parties win the sortition more than $(1+\varepsilon)fn$ times for a given committee.~~

H_4: Abort if any honest machine receives more than k nominations from honest parties between receiving Ready and the subsequent Delete.

H_5: Abort the experiment if the adversary ever broadcasts (cid, ek, vk, c, π) where more than one honest machine can decrypt n to something different to $\perp$.

H_6: For each KM-NCE encryption from an honest machine to a currently honest machine produce the ciphertext using the *faking* algorithm rather than encrypting. If the receiving machine is corrupted before it is given Delete the next time, the simulator will receive the secret keys for the role from $\mathcal{F}_{\mathsf{RA}}$. It may use the Open algorithm to produce an appropriate secret key, which correctly decrypts the faked ciphertexts.

H_7: Rather than choosing which honest machines have been nominated when the nomination ciphertext is generated, sample this lazily for committees where nomination is not finished. When the adversary corrupts a new machine, for each honest nomination in a committee assign the nomination to the machine with probability $1/|H|$. When a role is assigned to the machine input the corresponding $(\textsc{Corrupt-Nominee}, \ldots)$ to the functionality to receive secret keys for the role. Use the Open algorithm to generate the secret keys such that the machine can decrypt the corresponding ciphertext.

H_8: When an honest machine is given $(\textsc{New}, \mathsf{cid}, C)$, with probability s if the machine has won the sortition do the following:

- With probability $\dfrac{|H|}{|P|}$ input $(\textsc{Nom-Honest}, cid)$ to $\mathcal{F}_{\mathsf{RA}}$. Upon receiving $(\textsc{Nom-Honest}, cid, ek, vk)$ post (cid, ek, vk, c) to the bulletin board, where n is a faked ciphertext as previously.
- Otherwise, choose a uniformly random corrupt party $pid \leftarrow I$, and sample fresh keys giving input $(\textsc{Nom-Corrupt}, cid, pid, (ek, vk), (dk, sk))$ to $\mathcal{F}_{\mathsf{RA}}$.

H_9: We will now reintroduce the bounds on the number of honest and corrupt nominations, bounding total number corrupt nominations and nominee corruptions by $\mathcal{T}(|\mathcal{I}|/|\mathcal{P}|)$, ~~and the total number of nominations n~~ . The simulator aborts if too many nomination queries are made.

H_{10}: The simulator aborts if there are less than n nominations. We are now in the ideal world.

Indistinguishability of the hybrids. We now argue the adjacent hybrids are indistinguishable.

$H_0 \approx H_1$ Consider any environment $\mathcal{Z}$ and real-world adversary $\mathcal{A}$ that distinguish H_1 from H_2 with non-negligible advantage. We build a *TLP-hiding* adversary that, on each honestly generated puz, replaces transcript of communication with $\mathcal{F}_{\mathsf{TLP}}$ on $(\textsc{CreatePuzzle}, _, _, m)$ by transcript of communication on $(\textsc{CreatePuzzle}, _, _, (b\text{-msg})$ where $b \in \{0,1\}$ selects between m and $0^{|m|}$. Any distinguishing advantage prior to time $r+\delta$ yields a break of TLP hiding up to delay δ; after time $r+\delta$ both hybrids reveal the same m. Thus

$$\left|\Pr[out_{H_1} = 1] - \Pr[out_{H_2} = 1]\right| \leq \mathsf{negl}(\lambda).$$

$H_1 \approx H_2$ In H_2, honestly-generated ciphertexts already have the distribution tlp, which hides m. Replacing those with identically distributed, message-independent synthetic headers $\hat{c}$ leaves the wire view unchanged; openings occur at the same times and reveal the same pid.

$H_2 \approx H_3$ ~~An adversary winning the sortition too often, given the random seed from $\mathcal{F}_{\mathsf{Beacon}}$, may be reduced to an adversary breaking the pseudorandomness property of the sortition, by predicting which honest parties are going to win and adaptively corrupting them.~~

$H_3 \approx H_4$ The hybrid may only abort if there are at least k collisions between the honestly nominated roles. If the number of roles R is at least κ and $N = R^{2+c}$ for $c \geq 1$ this will only happen with negligible probability.

$H_4 \approx H_5$ The indistinguishability of these hybrids follows directly from the unique recipient property of the KM-NCE scheme.

$H_5 \approx H_6$ These hybrids are indistinguishable by a reduction to the KMNC-CCA security of the encryption scheme. Note, we need the CCA variant as we use the decryption oracle to recover messages sent to the honest parties, and input these to the ideal functionality. By our previous hybrids there are at most k honest nominations for the same set of honest keys, allowing us to stay within the constant number of challenges tolerated by the scheme. An adversary distinguishing the two hybrids would win the KMNC-CCA game with the same advantage.

$H_6 \approx H_7$ Sampling the assignment of roles to machines lazily induces the same probability distribution as if done beforehand.

$H_7 \approx H_8$ The honest machines sample the keys identically to $\mathcal{F}_{\mathsf{RA}}$, and nominate an honest party with probability $|H|/N$.

$H_8 \approx H_9$ We wish to upper bound the number Z of corrupt nominations in a committee.
Consider an adversary corrupting up to a fraction f of the machines. The faked nomination ciphertexts provide no information on which parties have been nominated. Corrupting a machine earlier does not change its probability of being nominated by honest parties, while the adversary gains the chance to control a nomination. Therefore, the best strategy for an adversary would be to expend the entire budget f at the beginning of the nomination process. This puts us in the situation analyzed in Section 4.2, and we can apply Equation (1) with $n = \omega(\log \lambda)$ and $\beta = \mathcal{T}(f)/c$ to conclude that, with overwhelming probability, the adversary will never exceed the corruption budget $\mathcal{T}$.

$H_9 \approx H_{10}$ We wish to ensure that there are at least n nominations. Note that the corrupt parties can always withhold nominations, so we need to guarantee that the honest parties themselves can fill the committee even if the dishonest sortition winners abort. Let $X \sim \mathsf{Bin}((1-f)N, s)$ be the random variable describing the number of winning draws of honest parties. Since $s = (1+\eta)/(1-f) \cdot c/N$, the expected number of winning draws in the first round is $\mathbb{E}[X] = (1-f)Ns = (1+\eta)n$. Therefore, by a Chernoff bound, we conclude that $\Pr[X < n] = \mathsf{negl}(\lambda)$ as long as $n = \omega(\log \lambda)$.

This completes the sequence of hybrids and the simulation. Hence the real and ideal executions are indistinguishable except with negligible advantage, and the protocol Π_{ERA} UC-realizes the functionality $\mathcal{F}_{\mathsf{ERA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{U}, 2, \delta)$ under the stated assumptions.

I NIZK relations.

I.1 Nomination relation $\mathcal{R}_{\mathsf{nom}}$.

In this section, we define the NIZK functionality proving the validity of nominations. In the real world, in which the TLP is instantiated with a circuit, one would simply define an appropriate NP relation $\mathcal{R}_{\mathsf{nom}}$. Unfortunately, this is not possible in our setting, as the TLP is defined only through an ideal functionality $\mathcal{F}_{\mathsf{TLP}}$, and we cannot do "NIZKs over functionality" in a sound way. Instead, we use joint-state UC [CR03] to define the NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$, abusing the notation and writing $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}]$ as if it were a standard NIZK functionality defined with respect to a relation. To make exposition more approachable, we will first define the relation $\mathcal{R}_{\mathsf{nom}}$ as if we were to instantiate the TLP with a circuit, and then we will explain how to use the global UC framework to make it rigorous.

Alternative approach (concrete TLP instantiation). Instead of modeling time-locking solely via the ideal functionality $\mathcal{F}_{\mathsf{TLP}}$ and letting TLPVer access its internal state, one can instantiate the time-lock puzzle with a concrete scheme $\mathsf{TLP} = (\mathsf{TLP}.\textsc{CreatePuzzle}, \mathsf{TLP}.\textsc{Solve}, \mathsf{TLP}.\textsc{GetMsg}, \mathsf{TLPVer})$. In this case the binding between the published puzzle tlp and the hidden value is captured by an explicit NP relation. Concretely, we extend the witness with the puzzle-creation randomness/opening τ and replace the time-lock check in $\mathcal{R}_{\mathsf{nom}}$ by

$$\mathsf{TLPVer}(\mathsf{tlp}, P^0.\mathsf{pk}, \Delta; \tau) = 1,$$

which asserts that tlp was honestly generated to hide $P^0.\mathsf{pk}$ for delay Δ using randomness τ (without solving tlp). Denote the resulting relation by $\mathcal{R}_{\mathsf{nom}}^{\mathsf{TLP}}$. It is a standard NP relation over public input $\mathbf{x} = (\mathsf{ek}, \mathsf{vk}, \mathsf{setP}, \mathsf{ct}, \mathsf{tlp}, \Delta)$ and witness $\mathbf{w} = (\mathsf{dk}, \mathsf{sk}, j, P^0.\mathsf{pk}, r_{\mathsf{ct}}, \tau)$. Consequently, in the $\mathcal{F}_{\mathsf{CR}}^{\star}$-hybrid one may use a standard UC-secure NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{nom}}^{\mathsf{TLP}}]$ and avoid "NIZKs over functionality." The tradeoff is that the proof (and, if desired, a separate UC realization argument) now depends on the chosen concrete instantiation of TLP.

Global UC. The Global UC (GUC) framework [CDPW07] extends the UC model by introducing *global functionalities*, i.e. idealized setup resources that persist across all protocol instances and can be accessed by any party or adversary. Typical examples include a CRS, a PKI, or global ledgers. A distinctive feature of GUC, compared to plain UC, is that *global functionalities may interact with each other*: they are not isolated boxes but can query one another and thus enforce joint invariants across protocols. This capability has been formalized and exploited in several works (e.g., [BDD+21b, CR03]). In our case, this means that the global time-lock functionality $\mathcal{F}_{\mathsf{TLP}}$ can internally invoke the global NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ to issue proofs of correct computation (e.g. that a published puzzle indeed commits to a value consistent with other protocol elements). This commit-and-prove linkage across global functionalities is crucial for ensuring consistency and UC-security in our construction.

Let TLP be a time-lock puzzle scheme with creator-side verification TLPVer that checks puzzle well-formedness given the creation randomness. The statement consists of:

(i) the public keys chosen by M: an encryption key ek and a signature verification key vk,
(ii) the public set of parties $\mathcal{P} = \{(\mathsf{pid}_i, \mathsf{pk}_i)\}_{i\in[N]}$,
(iii) a ciphertext ct (intended to encrypt the private keys for a chosen party $P' \in \mathcal{P}$),
(iv) a time-lock puzzle tlp and a delay parameter Δ (in rounds).

Formally,

$$\mathbf{x} = (\mathsf{ek},\ \mathsf{vk},\ \mathsf{setP},\ \mathsf{ct},\ \mathsf{tlp},\ \Delta)\,.$$

The witness consists of:

(i) the secret keys corresponding to the public keys: the decryption key dk and the signing key sk,
(ii) an index $j \in [N]$ and the chosen party's public key $P'.\mathsf{pk} = \mathsf{pk}_j$,
(iii) the encryption randomness r_{ct} used to produce ct,

Formally,

$$\mathbf{w} = (\mathsf{dk},\ \mathsf{sk},\ j,\ P'.\mathsf{pk},\ r_{\mathsf{ct}})\,.$$

The relation holds iff all of the following checks pass:

$$\begin{aligned}
&\textit{(keypair validity)} && \mathsf{KeyCheck}_{\mathsf{PKE}}(\mathsf{ek},\mathsf{dk}) = 1 \ \wedge\ \mathsf{KeyCheck}_{\mathsf{SIG}}(\mathsf{vk},\mathsf{sk}) = 1;\\
&\textit{(party selection)} && j \in [N] \ \wedge\ P'.\mathsf{pk} = \mathsf{pk}_j \ \wedge\ (\mathsf{pid}_j, \mathsf{pk}_j) \in \mathsf{setP};\\
&\textit{(encrypt private keys for } P'\textit{)} && \mathsf{ct} = \mathsf{KM{-}NCE.Enc}\,(P'.\mathsf{pk},\ (\mathsf{dk},\mathsf{sk});\ r_{\mathsf{ct}})\,;\\
&\textit{(time-lock binds } P'.\mathsf{pk} \textit{ and delay)} && \mathsf{TLPVer}\,(\mathsf{tlp},\ P'.\mathsf{pk},\ \Delta) = 1.
\end{aligned}$$

To make formal sense of the relation $\mathcal{R}_{\mathsf{nom}}$ in our setting, we use the global UC framework [CDPW07]. We let TLPVer access the internals of the functionality $\mathcal{F}_{\mathsf{TLP}}$. Given $\mathsf{tlp} = (\mathsf{st}_0, \Gamma, \mathtt{tag})$, TLPVer accesses the $\mathtt{omsg}$ list (which only the owner $\mathcal{P}_o$ can do), finds the tuple $(\mathsf{st}_0, \mathtt{tag}, \mathsf{st}_\Gamma, m)$ and checks that $m \in \mathsf{setP}$. In other words, $\mathcal{F}_{\mathsf{NIZK}}$ would check whether $\mathcal{R}_{\mathsf{nom}}(\mathbf{x}, \mathbf{w}) = 1$ by evaluating the corresponding circuit, in our case this is replaced by evaluating the circuit corresponding to everything except the TLP check *and* accessing the internals of $\mathcal{F}_{\mathsf{TLP}}$ to verify the TLP.

Remarks

1. $\mathsf{KeyCheck}_{\mathsf{PKE}}$ and $\mathsf{KeyCheck}_{\mathsf{SIG}}$ are the standard key-consistency predicates for the encryption and signature schemes (e.g., re-running keygen from seed or checking scheme-specific relations). To stay compatible with the protocol Π_{RA}, we assume that $\mathsf{KeyCheck}_{\mathsf{PKE}}(\bot, \bot) = 1$ and $\mathsf{KeyCheck}_{\mathsf{SIG}}(\bot, \bot) = 1$.
2. The time-lock check asserts that tlp was created to hide $P'.\mathsf{pk}$ with delay Δ using randomness τ; it does not require actually solving tlp (which would take at least Δ rounds).
3. Together, these constraints ensure that M (identified by $(\mathsf{pid}, M.\mathsf{pk})$) indeed *won* sortition for $(r, \mathsf{pid}, \mathsf{cid}, \rho)$, that M chose fresh key pairs $(\mathsf{ek}, \mathsf{dk})$ and $(\mathsf{vk}, \mathsf{sk})$, encrypted the private keys for some publicly listed party $P' \in \mathcal{P}$, and produced a time-lock puzzle that hides $P'.\mathsf{pk}$ until at least Δ rounds have elapsed.

I.2 YOSSO relation $\mathcal{R}_{\mathsf{YOSSO}}$

In this section, we describe the relation use to prove the correctness of the initial sharing and the resharing in our YOSSO protocol of Section 7. At a high level, in case of the initial sharing, the relation ensures that the ciphertexts indeed contain a valid sharing. In case of the resharing, the relation follows the resharing procedures, and ensures that the party correctly:(i) decrypted the ciphertexts it received from the previous committee, (ii) reconstructed the secret from the decrypted shares, (iii) reshared the secret, (iv) reencrypted the resharing under the ephemeral keys of the next committee. We now describe the relation in more detail.

The statement consists of:

(i) the secret sharing threshold t
(ii) the ciphertexts $e_1, \ldots, e_n$ that the party performing the role R received from the previous committee (possibly equal $\bot$ if initial sharing),
(iii) the ephemeral encryption keys $\mathsf{ek}'_1, \ldots, \mathsf{ek}'_n$ for the next committee,
(iv) the ciphertexts $e'_1, \ldots, e'_n$ addressed to the next committee.

Formally, the statement is $x = \left(t, (e_1, \ldots, e_n),\ (\mathsf{ek}'_1, \ldots, \mathsf{ek}'_n),\ (e'_1, \ldots, e'_n)\right)$. The witness consists of the following data:

(i) the decryption key dk that the party received from $\mathcal{F}_{\mathsf{ERA}}$,
(ii) the randomness ρ used to generate the resharing,
(iii) the encryption randomness $r'_1, \ldots, r'_n$ used to encrypt the resharing.

Formally, the witness is $w = (\mathsf{dk},\ \rho, S,\ r'_1, \ldots, r'_n)$, The relation holds iff either

$$
\begin{aligned}
(r > 0) \quad & s = \bot \\
\textit{(decrypt)} \quad & u_k \leftarrow \mathsf{Dec}(\mathsf{dk}, e_k) \text{ for } k \in S \\
\textit{(enough shares)} \quad & |S| \geq t; \\
\textit{(reconstruct)} \quad & u \leftarrow \mathsf{Reconstr}\left(\{(k, u_k)\}_{k \in S}\right); \\
\textit{(reshare)} \quad & (u'_1, \ldots, u'_n) \leftarrow \mathsf{Share}(u; \rho); \\
\textit{(re-encrypt)} \quad & \forall k \in [n] :\ e'_k = \mathsf{Enc}(\mathsf{ek}'_k, u'_k; r'_k).
\end{aligned}
$$

or

$$
\begin{aligned}
(r = 0) \quad & \mathsf{dk} = \bot, e_1 = \bot, \ldots, e_n = \bot, S = \bot \\
\textit{(reshare)} \quad & (u'_1, \ldots, u'_n) \leftarrow \mathsf{Share}(s; \rho); \\
\textit{(re-encrypt)} \quad & \forall k \in [n] :\ e'_k = \mathsf{Enc}(\mathsf{ek}'_k, u'_k; r'_k).
\end{aligned}
$$

Note that this already implies that the k-th ciphertext e'_k can be decrypted by the party owning the decryption key dk corresponding to ek'_k.

J Security proof of YOSSO

In this section we prove several properties of the YOSSO protocol of Section 7, such as secrecy, robust reconstruction under fraud proofs (defined below), and unframability. We introduce the following relaxation of the ECPSS robustness property.

Definition 1 (Robust reconstruction under fraud proofs). *After receiving their shares from an honest dealer, the reconstruction protocol run by at least t honest parties will output the correct secret σ (except with negligible probability).*
Moreover, either (1) the reconstruction protocol will either be executed, or (2) a publicly-verifable fraud proof will be produced, identifying all parties who did not perform the resharing correctly or aborted.

Since our scheme provides monetary incentives, we also need an additional requirement:

Definition 2 (Unframability). *No honest member of the holding committee is identified as misbehaving, except with negligible probability.*

J.1 Security analysis

The theorem below is a counterpart of [BGG+20, Theorem 3.1]. It crucially relies on the fact that $\mathcal{F}_{\mathsf{ERA}}$ always return a committee of size n.

Theorem 3. *Let $\mathcal{P}$ be the set of parties allowed to participate in the protocol, with $|\mathcal{P}| = N$, let $C = \Omega(\lambda)$ be the committee size, and let $t = T(f) = C(\varepsilon + f(2-f))$, for some constant $\varepsilon > 0$, be the committee threshold.*[6] *Let $\mathcal{F}_{\mathsf{ERA}} = \mathcal{F}_{\mathsf{ERA}}(\mathcal{P}, , T, \mathcal{U}, \mathtt{delay}, \delta)$ where $\delta = \mathtt{delay} + 1$.*

Then the protocol Π_{YOSSO} with threshold $t = T(f)$, is a scalable ECPSS scheme, satisfying secrecy and robust reconstruction under penalties, against PPT adaptive adversaries with corruption budget bounded by fN, for any $0 \leq f < 1$.

Proof. We prove the properties one by one.

Secrecy. The proof follows the lines of the secrecy proof of [BGG+20], adapted to our setting. In fact, since we use ideal functionalities for NIZKs and role assignment, the proof is significantly simpler. Consider an adversary that specifies two secrets s_0, s_1 and then interacts with our ECPSS scheme. We need to argue that it only has a negligible distinguishing advantage.

In the game, the challenger plays the role of all the honest parties and the (honest) dealer. In particular, it knows the global secret and all the initial shares. Since NIZKs are handled by the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$, the ciphertexts always contain a valid resharing, and zero-knowledge is trivially satisfied. Furthermore, since the resharing is always completed without one round, the time-lock puzzles created the nominations still remain hiding (which is, again, captured by the ideal functionality). Consequently, the role assignment functionality $\mathcal{F}_{\mathsf{ERA}}$ guarantees that the holding committee never contains more than $t = T(f)$ corrupted parties. This guarantees that the adversary's view is the same, regardless whether s_0 or s_1 is shared, due to the secrecy of the underlying PKE.

Robust reconstruction under penalties. Notice that the reconstruction always outputs the correct secret, if it occurs, thanks to the resharing proofs. Moreover, notice that the only way for the reconstruction not to occur is when there are not enough honest parties to reconstruct the resharing, which can only happen if the adversary corrupts enough parties to cause the resharing to fail. Therefore, at least $n - t + 1$ parties will be identified. Moreover, the fraud proofs produced by $\mathcal{F}_{\mathsf{ERA}}$ are publicly verifiable.

[6] This is equivalent to absorbing the constants of Theorem 2 into a single term.

Unframability. First of all, note that an honest party will never abort in **Execute-Role** without calling IdentifyNominee. This is true because the reconstruction in step 5 will always succeed (except with negligible probability) - this is guaranteed by the initial proof in round 1 and by the resharing proof in subsequent rounds. Furthermore, any party that identified as misbehaving will either not have sent its message to $\mathcal{F}_{\mathsf{BB}}$ (and an honest party always will) or will have sent a proof π that will not verify (and an honest party always sends a valid proof thanks to correctness of the NIZK). □

K General compiler.

Compared to the original YOSO compiler [CKR+23], the updated protocol $\mathsf{Compile}(\Pi)$ replaces the role assignment functionality $\mathcal{F}_{\mathsf{RA}}$ with its identifiable-abort extension $\mathcal{F}_{\mathsf{ERA}}$ and introduces minimal bookkeeping to support nominee identification upon abort. In particular, it maintains a map KeyCtx that records, for each non-dummy ephemeral public key ek, the corresponding nomination context (cid, r_{ctx}) needed to form well-typed $\mathcal{F}_{\mathsf{ERA}}.\mathsf{IdentifyNominee}(cid, r_{\mathrm{ctx}}, ek)$ queries. It also adds a per-session outbox Outbox^{sid} to bundle multiple signed role-messages into a single bulletin-board post, ensuring each machine posts at most once per (sid, r), and augments the read logic with Seen^{sid} and Msg^{sid} so that if a role speaks twice in the same (sid, r) then all of its messages for that round are ignored. Finally, when Π produces an identifiable-abort witness (e.g., a set of computation roles that were required to speak but did not), the compiler exports publicly verifiable abort evidence as a set of triples $\{(cid, r_{\mathrm{ctx}}, ek)\}$, allowing any party to invoke $\mathcal{F}_{\mathsf{ERA}}.\mathsf{IdentifyNominee}$ and obtain the corresponding nominee identifier(s) after the identification delay.

General compiler $\mathsf{Compile}(\Pi)$ **with identifiable aborts**

Notation: The algorithm $\mathsf{Nominate}(r, \{\mathsf{Broadcast}_{\mathsf{sid}}\}_{sid \in \mathsf{SID}})$ denotes a publicly computable function which when given the current round and public state outputs the set of committees to be nominated. We additionally maintain the following local bookkeeping (reconstructible by any party that invokes the same $\mathcal{F}_{\mathsf{ERA}}$ reads):

- KeyCtx: a map from ephemeral public keys to nomination context, where $\mathsf{KeyCtx}[ek] = (cid, r_{\mathrm{ctx}})$. This is used only to form well-typed $\mathsf{IdentifyNominee}$ queries upon abort. Here r_{ctx} is the same round-parameter used in the $\mathcal{F}_{\mathsf{ERA}}$ interfaces $(\mathsf{Read}, cid, r_{\mathrm{ctx}})$ and $(\mathsf{IdentifyNominee}, cid, r_{\mathrm{ctx}}, ek)$.
- Outbox^{sid}: a per-session bundle (list) of signed role-messages to be posted to $\mathcal{F}_{\mathsf{BB}}$ in round r (so each machine posts at most once per (sid, r)).
- $\mathsf{Seen}^{sid}[r, S] \in \{\bot, \mathsf{stored}, \mathsf{dup}\}$: used to enforce the rule "if a role speaks twice in the same (sid, r), ignore all of its messages".
- $\mathsf{Msg}^{sid}[r, S]$: stores the unique accepted message msg from sender S in (sid, r) (only defined if $\mathsf{Seen}^{sid}[r, S] = \mathsf{stored}$).

Init: Initialise sets of messages and keys for each role:

- For each $R \in \mathsf{Role}$ and $sid \in \mathsf{SID}$ define a set $R.\mathsf{Rec}^{sid} \leftarrow \emptyset$ of ciphertexts sent to the role. Set $R.\mathsf{ek} \leftarrow \bot$, $R.\mathsf{vk} \leftarrow \bot$, $R.\mathsf{dk} \leftarrow \bot$, and $R.\mathsf{sk} \leftarrow \bot$.
- If $R \in \mathsf{Role}^{\mathsf{In}} \cup \mathsf{Role}^{\mathsf{Out}}$, set $R.\mathsf{ek}$ and $R.\mathsf{vk}$ to relevant public keys established by PKI.
- For each $sid \in \mathsf{SID}$: $\mathsf{Broadcast}_{\mathsf{sid}} = \emptyset$.
- Initialise $\mathsf{KeyCtx} \leftarrow \emptyset$, and for each $sid \in \mathsf{SID}$ set $\mathsf{Outbox}^{sid} \leftarrow [\,]$, $\mathsf{Seen}^{sid} \leftarrow \emptyset$, and $\mathsf{Msg}^{sid} \leftarrow \emptyset$.

Nominate: In the beginning of round r, compute the (computation) committees to be nominated, $\{(cid_i, C_i)\}_{i \in [k]} \leftarrow \mathsf{Nominate}(r, \{\mathsf{Broadcast}_{\mathsf{sid}}\}_{sid \in \mathsf{SID}})$. For each committee input $(\mathsf{New}, cid_i, C_i)$ to $\mathcal{F}_{\mathsf{ERA}}$.

Role Keys: Once the machine finishes nominating committees in a round r, it proceeds to read the keys for the committees nominated at least `delay` round(s) earlier (as per the public schedule) . For each such committee, the machine inputs (Read, cid, r) to $\mathcal{F}_{\mathsf{ERA}}$ receiving lists PKeys and SKeys.

– For each element $(\mathsf{ek}, \mathsf{vk}, R_0)$ in PKeys the machine stores the role keys as $R_0.\mathsf{ek} \leftarrow \mathsf{ek}$ and $R_0.\mathsf{vk} \leftarrow \mathsf{vk}$, and if $\mathsf{ek} \neq \perp$ records $\mathsf{KeyCtx}[\mathsf{ek}] \leftarrow (cid, r)$.
– For each element $(pid, \mathsf{dk}, \mathsf{sk}, R)$ in SKeys (where pid corresponds to the machine's identifier) store the keys $R.\mathsf{dk} \leftarrow \mathsf{dk}$, $R.\mathsf{sk} \leftarrow \mathsf{sk}$. We now consider the machine to have been assigned role R.

Read: After storing new role keys each machine reads the bulletin board to process the next round of messages in the protocol. In round r the machine inputs $(\mathsf{Read}, sid, r-1)$ to $\mathcal{F}_{\mathsf{BB}}$; for each output element $(M', r', post')$ it receives the machine does the following:

– Parse $post'$ as a bundle $[msg'_1, \ldots, msg'_m]$ of signed role-messages (or treat a non-list $post'$ as the singleton bundle $[post']$).
– For each element msg'_ℓ in the bundle do:
  • Parse msg'_ℓ as (msg_ℓ, σ_ℓ) and parse msg_ℓ as $(S_\ell, r_\ell, sid, (R_1, x_1), \ldots, (R_k, x_k), x)$.
  • If $r_\ell \neq r-1$ then ignore msg'_ℓ.
  • Verify $b \leftarrow \mathsf{SIG}.\textsc{Verify}(S_\ell.\mathsf{vk}, msg_\ell, \sigma_\ell)$, ignoring msg'_ℓ if $b = 0$.
  • Enforce uniqueness per sender: If $\mathsf{Seen}^{sid}[r_\ell, S_\ell] = \perp$, then set $\mathsf{Seen}^{sid}[r_\ell, S_\ell] \leftarrow \mathsf{stored}$ and set $\mathsf{Msg}^{sid}[r_\ell, S_\ell] \leftarrow msg_\ell$. Else set $\mathsf{Seen}^{sid}[r_\ell, S_\ell] \leftarrow \mathsf{dup}$ and delete $\mathsf{Msg}^{sid}[r_\ell, S_\ell]$ (if defined), discarding *all* messages from S_ℓ for this (sid, r_ℓ).
– For each sender S with $\mathsf{Seen}^{sid}[r-1, S] = \mathsf{stored}$, commit its unique accepted message by:
  • Parse $\mathsf{Msg}^{sid}[r-1, S]$ as $(S, r-1, sid, (R_1, x_1), \ldots, (R_k, x_k), x)$.
  • Add (S, x) to $\mathsf{Broadcast}_{\mathsf{sid}}$.
  • For each $i \in [k]$ add (S, x_i) to $R_i.\mathsf{Rec}^{sid}$.

Role Execution: When a machine has been assigned a role R, it should run the role in its head and emulate the interaction between the role and its ideal functionality $\mathcal{F}^{sid}_{\mathsf{BC\&SPP}}$. In a given round a machine should activate each role it has been assigned, until the role signals that it has completed the round.

– If $R \in \mathsf{Role}^{\mathsf{In}}$, then this machine (belongs to $\mathsf{Machine}^{\mathsf{In}}$) must have received command (Input, x) which it passes on to R.
– If R inputs $(\mathsf{Read}, R, S, r')$ to $\mathcal{F}^{sid}_{\mathsf{BC\&SPP}}$, the machine should retrieve the tuple of the form (S, x) in $R.\mathsf{Rec}^{sid}$; if no such tuple exists, $\perp$ should be output directly to the role. The ciphertext should then be decrypted to obtain $x \leftarrow \mathsf{PKE.Dec}^{(sid,S)}(R.\mathsf{dk}, x)$ which may be returned to R.
– If R inputs (Read, S, r') to $\mathcal{F}^{sid}_{\mathsf{BC\&SPP}}$, the machine should retrieve the tuple of the form (S, x) in $\mathsf{Broadcast}_{\mathsf{sid}}$, and return x to R, returning $\perp$ if no such value exists.
– If $R \in \mathsf{Role}^{\mathsf{Out}}$ outputs (Output, y), output the same.

Identifiable abort handling: If R outputs $(\mathsf{Abort}, \mathcal{A})$ where $\mathcal{A}$ is the identifiable-abort witness of Π (e.g., a set of computation-role identifiers $R^\star$ that were required to speak in this (sid, r) but did not), then the machine derives the corresponding set of missing ephemeral keys $\mathcal{E} \leftarrow \{R^\star.\mathsf{ek} : R^\star \in \mathcal{A},\ R^\star.\mathsf{ek} \neq \perp\}$. For each $ek \in \mathcal{E}$, look up $(cid, r_{\mathrm{ctx}}) \leftarrow \mathsf{KeyCtx}[ek]$ (if undefined, drop ek), and output the public abort evidence $\big(\mathsf{Abort}, \{(cid, r_{\mathrm{ctx}}, ek)\}_{ek \in \mathcal{E}}\big)$. Any party can send $\mathsf{IdentifyNominee}(cid, r_{\mathrm{ctx}}, ek)$ to $\mathcal{F}_{\mathsf{ERA}}$ on these triples to obtain the nominee identifier(s).

Send BC&SPP: When the role $R \in \mathsf{Role}^{\mathsf{In}} \cup \mathsf{Role}^{\mathsf{Comp}}$ assigned to M outputs $(Send, R, ((R_1, x_1), \ldots, (R_k, x_k)), x)$ to $\mathcal{F}_{\mathsf{BC\&SPP}}$ with session identifier sid do the following:

1. For $j \in [k]$: $x_j \leftarrow \mathsf{PKE.Enc}^{(sid,R)}(R_j.\mathsf{ek}, x_j; \rho_j)$.
2. Let $msg = (R, r, sid, (R_1, x_1), \ldots, (R_k, x_k), x)$.
3. Compute $\sigma \leftarrow \mathsf{SIG.Sig}(R.\mathsf{sk}, msg)$ and set $msg' = (msg, \sigma)$.
4. If $R \in \mathsf{Role}^{\mathsf{Comp}}$:
   – Input $(Delete)$ to $\mathcal{F}_{\mathsf{ERA}}$.

– Erase all private local state associated with the role R, excluding (R, msg, σ). In particular this includes $R.\mathsf{dk}$, $R.\mathsf{sk}$ and the entire state of the copy of R the machine has been running in its head.

5. Append msg' to Outbox^{sid} (do not post immediately).
6. (Deferred BB write.) At the end of the machine's processing for round r, post the bundle Outbox^{sid} to the bulletin board for this (sid, r) (if non-empty), then reset $\mathsf{Outbox}^{sid} \leftarrow []$.
7. Input $(Ready)$ to $\mathcal{F}_{\mathsf{ERA}}$ if $R \in \mathsf{Role}^{\mathsf{Comp}}$.

If a machine has been assigned multiple roles, it activates them until they have all sent a message or completed the round, accumulating their signed messages in each Outbox^{sid} and posting at most once per (sid, r).

[a] this verification is not needed if $S \in \mathsf{Role}^{\mathsf{In}} \cup \mathsf{Role}^{\mathsf{Out}}$.
[b] Here, signatures can be avoided if $R \in \mathsf{Role}^{\mathsf{In}}$.

K.1 Security

In a full end-to-end treatment one could additionally relate the *real* role-assignment mechanism to an *ideal* role-assignment functionality (e.g., by proving that the mechanism UC-emulates $\mathcal{F}_{\mathsf{RA}}$). We do *not* pursue such an economic/rational analysis here. Instead, throughout this appendix we work directly in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{ERA}})$-hybrid model and treat $\mathcal{F}_{\mathsf{ERA}}$ as a given resource.

Under this modeling choice, the compiler $\mathsf{Compile}(\Pi)$ remains structurally unchanged, except for the way it handles aborts. Namely, in the identifiable-abort variant, whenever a party detects that a message expected from some role (identified by an ephemeral public key ek) is missing, it invokes $\mathsf{IdentifyNominee}(cid, r_{\mathrm{ctx}}, ek)$ to obtain the corresponding nominee identifier, which is released only after the prescribed delay δ. This identifier (together with the publicly verifiable transcript evidence of the missing message) can then be used by an external judge to punish the misbehaving party.

Accordingly, Theorem 3 is strengthened as follows: assuming the abstract protocol Π UC-realizes $\mathcal{F}$ *with identifiable abort*, the compiled protocol $\mathsf{Compile}(\Pi)$ UC-realizes $\mathcal{F}$ *with identifiable abort* in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{ERA}})$-hybrid model. Changes to the original definitions are marked in gray.

Theorem 4 (General compiler with identifiable aborts). *Consider an abstract protocol Π which YOSO securely implements the ideal functionality $\mathcal{F}$ with identifiable abort in the presence of n/w static corruptions (where $n = \Omega(\kappa)$ denotes the committee size) in the $\mathcal{F}_{\mathsf{BC\&SPP}}$-hybrid model. Let $R_{\max} \geq \kappa$ denote the upper bound on the concurrently active roles at any point (which refers to roles that are able to receive messages, or currently being nominated). Let $\mathcal{F}_{\mathsf{ERA}}$ be shorthand for $\mathcal{F}_{\mathsf{ERA}}(P, c, T, U, 2, \delta_{\mathrm{IA}})$ where U samples the uniform distribution and*

$$T(f) = n\big(1 - (1-\epsilon)(1-f)^2\big),$$

for $\epsilon > 0$ and $\delta_{\mathrm{IA}} \geq 2$. Suppose Π is secure against an arbitrary number of static corruptions in the input and output roles and has the following properties

– *All honest roles in the same committee speak in the same round.*
– *It is publicly computable which committees need to be nominated at least* `delay` *round(s) in advance.*

Then the protocol $\mathsf{Compile}(\Pi)$ UC implements the ideal functionality $\mathcal{F}$ with identifiable abort in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{ERA}})$-hybrid model, under the presence of $T < N f_t$ adaptive corruptions of the computation machines and any number of static corruptions in the input and output roles, where $N = R_{\max}^{2+\delta}$ for a constant $1 \leq \delta$ and $0 < 1 - 2wf_t + wf_t^2$

Moreover, whenever $\mathsf{Compile}(\Pi)$ aborts because some computation role with ephemeral public key ek failed to speak in its designated round, any party can invoke $\mathcal{F}_{\mathsf{ERA}}.\mathsf{IdentifyNominee}(cid, r, ek)$ and, after δ_{IA} rounds, obtain the identifier pid of the machine that was nominated for that role.