Taming Adaptivity in YOSO Protocols: The Modular Way

Ran Canetti[1], Sebastian Kolby[2], Divya Ravi[2], Eduardo Soria-Vazquez[3], and Sophia Yakoubov[2]

[1] Boston University, USA; canetti@bu.edu
[2] Aarhus University, Denmark; {sk, divya, sophia.yakoubov}@cs.au.dk
[3] Technology Innovation Institute, UAE; eduardo.soria-vazquez@tii.ae

Abstract. YOSO-style MPC protocols (Gentry *et al.*, Crypto'21), are a promising framework where the overall computation is partitioned into small, short-lived pieces, delegated to subsets of one-time stateless parties. Such protocols enable gaining from the security benefits provided by using a large community of participants where "mass corruption" of a large fraction of participants is considered unlikely, while keeping the computational and communication costs manageable. However, fully realizing and analyzing YOSO-style protocols has proven to be challenging: While different components have been defined and realized in various works, there is a dearth of protocols that have reasonable efficiency and enjoy full end to end security against adaptive adversaries.

The YOSO model separates the protocol design, specifying the short-lived responsibilities, from the mechanisms assigning these responsibilities to machines participating in the computation. These protocol designs must then be translated to run directly on the machines, while preserving security guarantees. We provide a versatile and modular framework for analyzing the security of YOSO-style protocols, and show how to use it to compile any protocol design that is secure against *static* corruptions of t out of c parties, into protocols that withstand *adaptive* corruption of T out of N machines (where T/N is closely related to t/c, specifically when $t/c < 0.5$, we tolerate $T/N \leq 0.29$) at overall communication cost that is comparable to that of the traditional protocol even when $c << N$. Furthermore, we demonstrate how to minimize the use of costly non-committing encryption, thereby keeping the computational and communication overhead manageable even in practical terms, while still providing end to end security analysis. Combined with existing approaches for transforming stateful protocols into stateless ones while preserving static security (e.g. Gentry et al. 21, Kolby et al. 22), we obtain end to end security.

1 Introduction

Secure multiparty computation (MPC) allows data owners to outsource the processing of their sensitive data to a set of machines, with the guarantee that as long as fewer than a threshold t of those machines are corrupt, no-one will learn more about the data than revealed by the computation output. YOSO MPC [GHK+21] is an emerging new style of MPC where participating machines have very short term roles: they receive messages, performing an internal computation, and send messages in a single communication round to the next set of participating machines. Before sending those messages, the machine erases all other state relevant to the protocol execution.

The advantage of YOSO MPC is that the communication complexity of the protocol can be sublinear in N (the number of available machines), even if the corruption threshold T is linear in N. This might appear impossible, since if the communication complexity is sublinear in N, the set of all machines ever to send a message fits within the adversary's corruption budget; however, the crucial insight is that as long as an adversary cannot predict which machines will "speak", she is unable to target them. One of the challenges of YOSO MPC is choosing participating machines in an unpredictable way, making it harder to locate and adaptively attack those machines while they are active and relevant to the protocol.

YOSO MPC protocols naturally decompose into two tasks. The first of these is *role assignment*, which entails determining which machines will have a role to play and handing them the secret keys they will need in order to do so, while keeping their identities hidden from the adversary. The second task is actually running the MPC by having the chosen machines play their assigned roles.

One can view YOSO MPC protocols through two lenses: In the *natural world*, a protocol must specify instructions for physical machines, including instructions for role assignment; i.e., how the machines should go about determining whether they have a role to play, and if so, which one. In the *abstract world*, a YOSO MPC protocol can be described in terms of the roles alone, without consideration for the machines running them.

Some previous YOSO protocols (e.g. the protocol of Benhamouda *et al.* [BGG+20]) are described in the natural world, running both role assignment and computation in an entwined way. Others (e.g. the protocols of Gentry *et al.* [GHK+21] and Acharya *et al.* [AHKP22]) are described in

the abstract world, relying on behind-the-scenes machinery to take care of role assignment.

The second is a more modular approach, resulting in simpler protocol descriptions. However, these descriptions do not suffice for use in the real, natural world. We need a *compiler* to translate them into something machines can run; such a compiler might access an ideal role assignment functionality.

One such role assignment functionality and compiler were introduced by Gentry *et al.* [GHK+21]. However, the role assignment functionality presented by Gentry *et al.* was perhaps too strong, in that it did not allow the adversary to influence the role assignment, instead choosing *all* machines in an ideal, random way. This makes it impossible for the most efficient known role assignment mechanism (that of Benhamouda *et al.* [BGG+20]) to realize this functionality. Furthermore, the compiler of Gentry *et al.* [GHK+21] has two drawbacks: (a) it is inefficient, and (b) it is incompatible with some abstract protocols (e.g. the protocol of Braun *et al.* [BDO22] and Kolby *et al.* [KRY22]).

1.1 Our Contributions

In this paper, we fill the above gaps: we introduce a more realistic role assignment ideal functionality $\mathcal{F}_{\mathsf{RA}}$, give a realization of $\mathcal{F}_{\mathsf{RA}}$, and present a more efficient, more general compiler that relies on this new functionality. In particular, we use non-committing encryption only for implementing $\mathcal{F}_{\mathsf{RA}}$. All the messages of the underlying (statically secure) protocol are encrypted using standard (CCA secure) encryption.

1.1.1 Ideal Role Assignment Functionality In Section 3, we introduce our role assignment ideal functionality $\mathcal{F}_{\mathsf{RA}}$. Our goal is to capture a more general and broad class of potential and existing role assignment protocols. Towards this, we give a comprehensive design of $\mathcal{F}_{\mathsf{RA}}$ that supports modeling various assignment approaches.

At a very high-level $\mathcal{F}_{\mathsf{RA}}$ supports two kinds of elections: assignment of a role to an honest machine, and assignment influenced by the adversary, to a chosen, possibly corrupt machine. The machines are allowed to probe the $\mathcal{F}_{\mathsf{RA}}$ to read the public keys of the roles assigned so far, deduce if they themselves have been assigned a role, and retrieve the secret keys in such a case. Furthermore, our design of $\mathcal{F}_{\mathsf{RA}}$ supports modeling various scenarios that can occur during its execution, such as (a) when the adaptive adversary manages to corrupt a role that was assigned when it was

uncorrupted (before the election of the committee was completed), (b) when a machine finishes preparing her one-time message corresponding to a role and wishes to delete her state, and (c) when a machine who was previously engaged in executing a role is now ready to be assigned a new role. The formal details appear in Section 3.

1.1.2 Compiling Abstract Protocols In Section 4, we describe how to leverage $\mathcal{F}_{\mathsf{RA}}$ to compile an MPC protocol in the abstract world into one that can be run in the natural world. Unlike the compiler of Gentry *et al.* [GHK+21], we only use non-committing encryption within the realization of $\mathcal{F}_{\mathsf{RA}}$ (and not within the compiler itself). This has a two-fold advantage: (a) it yields a significant efficiency gain, and (b) it gives compatibility with a broader class of abstract YOSO protocols (e.g. the protocol of Braun *et al.* [BDO22] and Kolby *et al.* [KRY22]).

At a high-level, in our compiled protocol in the natural world, each machine deduces if it has been selected for a role by invoking the $\mathcal{F}_{\mathsf{RA}}$. If this is the case, it reads the bulletin board (in the natural world) to obtain ciphertexts encrypted using that role's public key. It can decrypt these ciphertexts using the secret keys provided by $\mathcal{F}_{\mathsf{RA}}$ and proceed to compute the outgoing messages of the role to other roles. These outgoing messages can be encrypted using the other roles' public keys (provided by $\mathcal{F}_{\mathsf{RA}}$) and posted on the bulletin board.

The main challenge is proving adaptive security of the compiled protocol, assuming that the underlying abstract protocol is only statically secure. The crux of our proof is that the set of corrupt roles can be chosen statically, and then the $\mathcal{F}_{\mathsf{RA}}$ can be suitably re-programmed so that adaptive corruption of machines can be appropriately matched to the already chosen static corrupt roles. We refer to Section 5 for details on the technicalities in our proof.

Compiling Abstract Protocols that Require Message Verification. The above compiler supports abstract protocols using only ideal point-to-point and broadcast channels. We note that this does not cover a large class of existing YOSO protocols that assume explicit access to keys for the roles to allow zero-knowledge proofs or any other types of public verifiability for point-to-point messages. In Section 6, we show how our compiler can be extended to abstract protocols that require such verification. More specifically, we modify the above compiler to accommodate abstract protocols that leverage the functionality $\mathcal{F}_{\mathsf{VeSPa}}$ [KRY22], which is used to enable

parties to prove to others that the broadcast and peer-to-peer messages they send within a protocol were derived honestly.

In order to extend our compiler to abstract protocols using $\mathcal{F}_{\mathsf{VeSPa}}$, we need to be able to emulate the verifiability of messages in the natural world. For this, we simply rely on augmenting the messages posted on the bulletin board in the compiled protocol with corresponding non-interactive zero-knowledge proofs proving that these messages were computed correctly.

1.1.3 Realizing the role assignment functionality In Section 7, we modify the role assignment protocol of Benhamouda *et al.* [BGG+20] to realize $\mathcal{F}_{\mathsf{RA}}$. As shown in [HLH+22], their protocol had problems in addressing the adaptivity of the adversary when it came to realizing the necessary anonymity property. As in [BGG+20], our modified protocol Π_{RA} uses a cryptographic sortition algorithm in order to ensure that an adversary is not able to increase the likelihood of corrupting a role of his choice. Furthermore, Π_{RA} uses Key and Message Non-Commiting Encryption (KM-NCE). This enables the simulator to deal with the different problematic scenarios described above. That is, by creating "fake" ciphertexts, the simulator can deal with the case of honest parties sending messages to recipients who were *a priori* expected to be honest, but then become corrupted by the adversary.

Crucially, our protocol instructs nominated machines to *erase* their private decryption key before making themselves known. As soon as the machine completes its role as a committee member, it chooses a new key pair and registers the new public encryption key with the PKI server. The machine will keep a (truly) long-term signature key in order to authenticate itself to the PKI server.

The much less efficient role assignment protocol of Gentry *et al.* [GHM+21] (which uses any MPC protocol to run random-index PIR) may be modified to trivially realize $\mathcal{F}_{\mathsf{RA}}$, by a similar application of KM-NCE.

2 Preliminaries

2.1 Key and Message Non-Commiting Encryption

We recall the notion of a Key and Message Non-Commiting Encryption (KM-NCE) from [HLH+22], which is an extension of receiver non-commiting encryption. Informally, a KM-NCE is a public-key encryption scheme that allows to generate fake ciphertexts *without any public key*

in such a way that those fake ciphertexts can later be decrypted to any plaintext by generating an appropriate secret key on the fly.

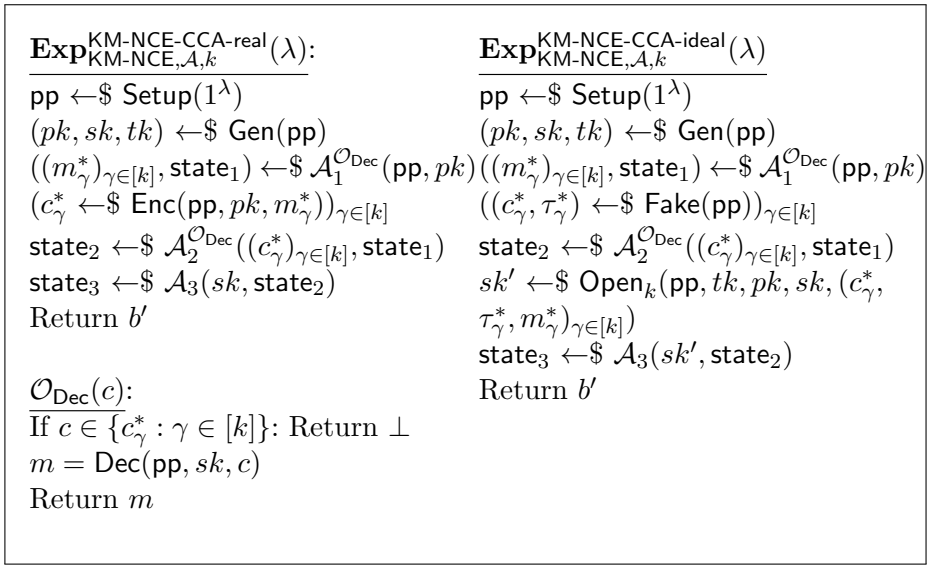

Fig. 1: The experiments for KM-NCE-CCA security of a KM-NCE scheme.

Definition 1 (Security). *A KM-NCE scheme* $\mathsf{KM\text{-}NCE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Fake}, \mathsf{Open}_k)$ *in the* k*-challenge setting is (*CCA-*)*secure *if for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, the advantage* $\mathbf{Adv}^{\mathsf{AC\text{-}RSO}_{k,\mathsf{Dist}}\&\mathsf{C\text{-}CPA}}_{\mathsf{PKE},\mathcal{A},\mathcal{S},\mathcal{D},n,t,k}(\lambda) :=$

$$|\Pr[\mathbf{Exp}^{\mathsf{KM\text{-}NCE\text{-}CCA\text{-}real}}_{\mathsf{KM\text{-}NCE},\mathcal{A},k}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{KM\text{-}NCE\text{-}CCA\text{-}ideal}}_{\mathsf{KM\text{-}NCE},\mathcal{A},k}(\lambda) = 1]|$$

is negligible, where $\mathbf{Exp}^{\mathsf{KM\text{-}NCE\text{-}CCA\text{-}real}}_{\mathsf{KM\text{-}NCE},\mathcal{A},k}$ *and* $\mathbf{Exp}^{\mathsf{KM\text{-}NCE\text{-}CCA\text{-}ideal}}_{\mathsf{KM\text{-}NCE},\mathcal{A},k}$ *are defined in Figure 1.*

KM-NCE schemes can be constructed from hash proof systems, as shown in [HLH+22].

2.1.1 KM-NCE with a unique recipient

We need to define an additional property for KM-NCE, which ensures that the adversary cannot produce (something that looks like) a ciphertext which decrypts under two different honest secret keys.

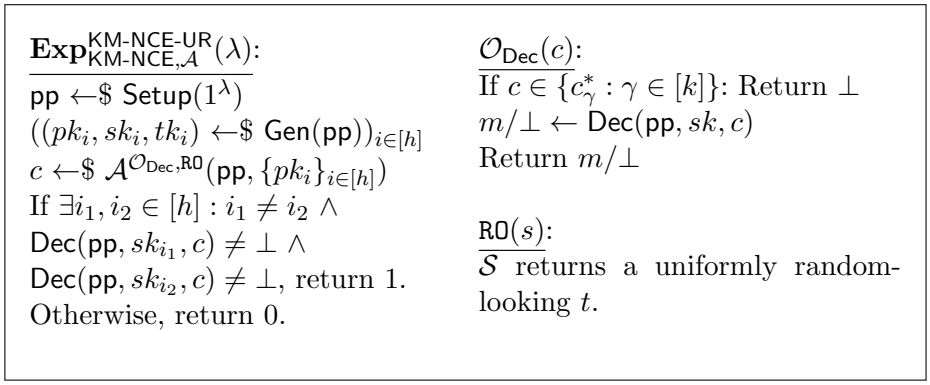

Fig. 2: The unique recipient experiment.

Definition 2 (Unique recipient). *A KM-NCE scheme* $\mathsf{KM\text{-}NCE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Fake}, \mathsf{Open}_k)$ *is* unique recipients *if for any* PPT *adversary* $\mathcal{A}$, $\Pr[\mathbf{Exp}^{\mathsf{KM\text{-}NCE\text{-}UR}}_{\mathsf{KM\text{-}NCE},\mathcal{A}}(\lambda) = 1]$ *is negligible, where* $\mathbf{Exp}^{\mathsf{KM\text{-}NCE\text{-}UR}}_{\mathsf{KM\text{-}NCE},\mathcal{A}}$ *is defined in Figure 2.*

2.1.2 A unique recipient KM-NCE construction We show how to build a unique recipient KM-NCE encryption scheme in the programmable random oracle model. Since this implies the notion of receiver non-committing encryption, we know that random oracles are necessary in order to avoid secret keys that are as long as the messages to be encrypted [Nie02].

Our construction is based on a simple variant of ElGamal, which makes it more efficient than the KM-NCE construction based on hash proof systems (HPS) from [HLH+22, Section 5.3], which relies on a matrix variant of DDH [EHK+13]. Furthermore, that construction does *not* have the unique recipient property that we need. The reason behind this is that, since the projected and unprojected hash need to coincide for elements x of the language, the adversary can use the unprojected hash (in their specific notation, $\widetilde{\mathtt{Pub}}$) together with the public keys of honest parties in order to try and find a suitable witness that leads to a collision (in their notation, the same $\widetilde{\pi}$) with several secret keys. Once he has that, it is easy for him to come up with the rest of the elements of the ciphertext (given x, any d can be fixed by varying the message m. Hence, a whole range of values $\tau = H(x, d)$ can be explored by the adversary). It is very easy for the adversary to come up with elements of the language x and their witnesses w, since this is a necessary feature for the practical efficiency of the encryption algorithm. Thus, we cannot rule out

maliciously created ciphertexts that decrypt to several recipients. In more detail, for the HPSs from [HLH+22, Section 6], each public key defines a hyperplane, and collisions happen at the intersection of any two such hyperplanes. This gives plenty of candidates for collisions.

Whereas the prior attack to the unique recipient property is specific to the instantiation of construction of [HLH+22, Section 5.3] with the HPSs from [HLH+22, Section 6], it is likely that similar attacks could be mounted for other natural constructions based on HPSs. The necessary relation between the public and private hash functions, together with any nice algebraic description of the public hashing algorithm (e.g. defining hyperplanes as in the attack above) would potentially lead to the same problem.

We define below our candidate construction based on a modification of ElGamal. The algorithms of our scheme are oracle algorithms with query access to the oracle $\mathtt{RO} : \{0,1\}^* \to \{0,1\}^{2\kappa}$, we let this be implicit in our notation.

- $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\kappa)$: Pick a cyclic group $\mathbb{G}$ of order q, where q is a κ-bit prime, and let g be a generator of $\mathbb{G}$. Let the message space of the encryption scheme be $\{0,1\}^\kappa$. Set $\mathsf{pp} = (\mathbb{G}, g, q)$ public parameters.
- $(pk, sk, \emptyset) \leftarrow\$ \mathsf{Gen}(\mathsf{pp})$: Sample $a \leftarrow\$ \mathbb{Z}_q$, let $sk = a$. Compute the public key $pk \leftarrow g^a$ and output $(pk, sk, \emptyset)$.
- $c \leftarrow\$ \mathsf{Enc}(\mathsf{pp}, pk, m)$: Sample $r \leftarrow\$ \mathbb{Z}_q$ and compute $\beta \leftarrow g^r$. Query the oracle for a mask $k \leftarrow \mathtt{RO}(pk^r)$ and a MAC $d \leftarrow \mathtt{RO}(r, m)$. Let $e \leftarrow k \oplus (r, m)$, and output $c = (\beta, e, d)$.
- $m \leftarrow \mathsf{Dec}(\mathsf{pp}, sk, c)$: Parse $c = (\beta, e, d)$. Query the oracle $k' \leftarrow \mathtt{RO}(\beta^{sk})$, compute $(r', m') \leftarrow e \oplus k'$. Check if $g^{r'} = \beta$ and $d = \mathtt{RO}(r', m')$, output m' if both conditions are satisfied, otherwise output $\perp$.
- $(c, \tau) \leftarrow\$ \mathsf{Fake}(\mathsf{pp})$: Sample $r \leftarrow\$ \mathbb{Z}_q$ and compute $\beta \leftarrow g^r$. Let $\tau = r$. Sample uniformly random strings $e, d \in \{0,1\}^{2\kappa}$ and let the fake ciphertext be $c = (\beta, e, d)$. Output (c, τ).
- $sk' \leftarrow \mathsf{Open}_k(\mathsf{pp}, pk, sk, (c^*_\gamma, \tau^*_\gamma, m^*_\gamma)_{\gamma \in [k]})$: To open a fake ciphertext $c^*_\gamma = (\beta, e, d)$ as an encryption a message m^*_γ to a chosen pk. Let $r = \tau^*_\gamma$, program the random oracle such that $\mathtt{RO}(r, m^*_\gamma) = d$ and $\mathtt{RO}(pk^r) = e \oplus (r, m^*_\gamma)$. Output $sk' = sk$.

Intuitively it is possible to replace ciphertexts by fakes as long as the adversary is unable to query either pk^r or (r, m) to the random oracle, if the adversary does query this values it may be used to solve the Diffie-Hellman problem. Including $d = \mathtt{RO}(r, m)$ allows the decryption oracle

to extract the plaintext and verify the integrity of the ciphertext without use of the secret key. We now formally prove the security of our KM-NCE scheme.

Theorem 1. *The construction above is* $\mathsf{KM\text{-}NCE}_k\text{-}\mathsf{CCA}$ *and unique recipient secure, in the pROM under the CDH assumption in group* $\mathbb{G}$.

Proof. First, we consider unique recipient security. Assume for contradiction there have been no collisions in random oracle, for a sufficiently large range and bounded adversary this holds with overwhelming probability. A winning adversary outputs a ciphertext $c = (\beta, e, d)$ such that for some sk_i, sk_j: $\mathsf{Dec}(\mathsf{pp}, sk_i, c) \neq \perp$ and $\mathsf{Dec}(\mathsf{pp}, sk_j, c) \neq \perp$. We subscript intermediate values in each decryption with the index of the secret key. For honestly generated keys $sk_i \neq sk_j$ with overwhelming probability, implying $\beta^{sk_i} \neq \beta^{sk_j}$. As a result, $k'_i \neq k'_j$ if there have been no collisions in the random oracle. This in turn implies that $(r'_i, n'_i) \neq (r'_j, n'_j)$. For both outputs to be different from $\perp$, it must be the case that $d = \mathtt{RO}(r'_i, n'_i) = \mathtt{RO}(r'_j, n'_j)$ raising a contradiction.

Now consider $\mathsf{KM\text{-}NCE}_k\text{-}\mathsf{CCA}$ security. Through a series of hybrids we will replace $c^*_\gamma = (\beta, e, d)$ with a fake ciphertext for each $\gamma \in [k]$. Faking a ciphertext is only different in how c and d are chosen. These two cases are only different in the oracle output on inputs pk^r and (r, m) prior to $\mathcal{A}_3$ receiving the secret key sk.

In the real and ideal worlds the adversary receives the same secret key sk and has access to an identically distributed random oracle. The only input which may differ is state_2, produced by $\mathcal{A}_2$. The views of Adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ only differ between the real and ideal game when querying pk^r or (r, m) to the random oracle. Thus, if $\mathcal{A}_3$ distinguishes the real and ideal worlds with non-negligible advantage then one of $\mathcal{A}_1, \mathcal{A}_2$ must query pk^r or (r, m) with probability equal to the advantage. We will argue that such a pair $(\mathcal{A}_1, \mathcal{A}_2)$ may be reduced to an adversary solving the computational Diffie-Hellman problem.

Consider an adversary which queries either pk^r or (r, m) with probability ϵ, while making at most t random oracle queries. Given a computational Diffie-Hellman instance $(g, x = g^a, y = g^r)$, we set $pk = x$ and $\beta = y$. Note, the solution to this instance is $pk^r = \beta^a$. We will address how to provide a decryption oracle without knowing the secret key a later. The reduction chooses a query index $i \leftarrow\$\ [t]$. When the adversary makes the ith query, if the input is of the form (r, m), the reduction outputs pk^r, if the input only consists of a single element z the reduction outputs this

directly. The reduction aborts before providing $\mathcal{A}_3$ the secret key. Note, the reduction needs $\tau = r$, which it does not have, to open the ciphertexts to $\mathcal{A}_3$, preventing the use of $\mathcal{A}_3$ in the reduction. The reduction yields an adversary solving the Diffie-Hellman problem with probability ϵ/t.

We now return to the issue of providing a suitable decryption oracle during our hybrids. Consider a ciphertext $c^* = (\beta^*, e^*, d^*)$ queried to the decryption oracle, which is not equal to any of the challenge ciphertexts. If d^* is not a random oracle output on an input of the form (r, m) output $\perp$, this includes any d for faked ciphertexts. A ciphertext using d from a challenge with $\beta^* \neq \beta$ or $e^* \neq e$, real decryption would result in $\perp$ with overwhelming probability.

For a given ciphertext, e and $k' = \mathtt{RO}(\beta^{sk})$ uniquely determine (r, m); if this has not yet been queried the probability $\mathtt{RO}(r, m) = d$ is $2^{-2\kappa}$, and we may safely return $\perp$. If d is an output of the random oracle the reduction may retrieve the corresponding input (r, m). We check if $\beta = g^r$, returning $\perp$ if this is not the case. Given r the oracle then computes $k' \leftarrow \mathtt{RO}(pk^r)$; $(r', m') \leftarrow e \oplus k'$. If $(r', m') = (r, m)$ output m, otherwise output $\perp$.

2.2 Cryptographic sortition

A cryptographic sortition protocol [CM19] allows to provably select a random subset of parties according to some timely and truthful randomness source such as Verifiable Random Functions (VRF) [MRV99]. Importantly, a party can find out whether it was selected through local computation, once it has received the output from the VRF. We will use the syntax $(\mathsf{draw}, \pi) \leftarrow \mathsf{Sortition}(sk^{\mathsf{VRF}}, (r, \mathsf{pid}, \mathsf{cid}, \rho))$, where sk^{VRF} is the secret key for a Verifiable Random Function, ρ is randomness drawn from a beacon and r, pid and cid will act as a separation tag among the different times the sortition algorithm will be necessary.

2.3 The You-Only-Speak-Once model

The YOSO model introduced by Gentry *et al.* [GHK+21] formalised a variant of the UC framework enabling the design of protocols focusing only on role execution, and not the mechanisms for role assignment or receiver anonymous communication. We will refer to protocols in this model as *abstract* YOSO protocols.

The YOSO model deviates from standard UC in the following ways:

- Parties are entirely replaced by roles, these roles are executed in the same way as UC parties, but are conceptually distinct in that they

do not map to any physical machine. Rather, roles represent abstract responsibilities which in a natural execution of a protocol would be carried out by the machine to which they were assigned to on the fly. Protocol design is indifferent to which actual machines would be executing the role.
- Idealised communication functionalities are provided to the roles executing a protocol, allowing point-to-point messages between roles. This corresponds to the availability of receiver anonymous communication channels, but ignores their realisation.
- Security is proven for "yosoified" versions of the protocol, where all roles are placed within a YOSO wrapper. This wrapper enforces that roles only speak once by killing them once they use a communication functionality. This is modelled by a SPOKE token which the ideal communication functionalities return upon the sending of messages. When receiving SPOKE the wrapper additionally forwards this to any sub-routines and its environment. Killing a role represents the machine running a role erasing any associated state, preventing the adversary from later corrupting the role.
- While we want natural YOSO protocols to be secure against an adaptive adversary, allowing the adversary this power in the abstract world would make protocol design significantly more difficult. Gentry *et al.* [GHK+21] make the observation that an adversary does not know which roles are assigned to a machine before it is corrupted. As a result the adversary may be restricted in the abstract world, while still being able to achieve adaptive security when translated to the natural world. This is enforced through a new "corruption controller" entity which dictates the types of corruptions the environment is allowed to make.

We recall the ideal functionality allowing point-to-point and broadcast communication as in [GHK+21]. Note, when a role S inputs the SEND it finally receives the SPOKE token, which will cause its wrapper to kill it.

Functionality $\mathcal{F}_{\mathsf{BC\&SPP}}$

This ideal functionality has the following behaviour:

- Initially create point-to-point and broadcast maps:
  $y : \mathbb{N} \times \mathsf{Role} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ where $y(r, \mathsf{R}, \mathsf{R}') = \perp$ for all $r, \mathsf{R}, \mathsf{R}'$
  $m : \mathbb{N} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ where $m(r, \mathsf{R}) = \perp$ for all r, R.
- On input $(\text{SEND}, \mathsf{S}, ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k)), x)$ in round r proceed as follows:

    - For $i \in [n]$ update $y(r, \mathsf{S}, \mathsf{R}_i) = x_i$. *Store point to point messages from the role.*
    - Update $m(r, \mathsf{S}) = x$. *Store the broadcast message from the role.*
    - Output $(\mathsf{S}, ((\mathsf{R}_1, |x_1|), \ldots, (\mathsf{R}_k, |x_k|)), x)$ to $\mathcal{S}$.
    - For corrupt roles R_i output x_i to $\mathcal{S}$. *Leak messages lengths and the broadcast message to the simulator in a rushing fashion.*
    - If S is honest give SPOKE to S.
- On input $(\text{READ}, \mathsf{R}, \mathsf{S}, r')$ in round r where $r' < r$ for $x = y(r', \mathsf{S}, \mathsf{R})$ output x to R.
- On input $(\text{READ}, \mathsf{S}, r')$ in round r where $r' < r$ output $x = m(r', \mathsf{S})$ to R.

The central paradigm of synchronous abstract YOSO protocols is that executions proceeds by a sequence of committees, each permitting a certain corruption threshold. These committees may potentially receive messages concurrently, or even speak in the same round.

2.4 Compiling abstract YOSO protocols

By their nature, protocols designed in the abstract YOSO model cannot be run directly on machines, they first have to undergo translation, or *compilation*, to the natural world.

This compilation reraises the issues of role assignment and receiver anonymous communication. Any compiler must provide equivalent guarantees of secure communication between roles in the protocol.

In their presentation of the YOSO model Gentry *et al.* [GHK+21] provide an example of compilation from the abstract to natural world. Their approach used a simplified toy timed ledger with role assignment functionality as a building block. This functionality provided the necessary keys for roles, which were then used to wrap messages in the underlying protocol in encryption. The compiler allowed the compilation of an abstract protocol secure against random adaptive point corruptions (i.e. an adversary only allowed to corrupt random roles), to a natural protocol secure against chosen adaptive point corruptions.

The focus of the compiler of Gentry *et al.* [GHK+21] was demonstrating the feasibility of compilation. As a result the compiler has a number of limitations, such as the role assignment functionality not being realised. Additionally, to achieve adaptive security the compiler uses

non-committing encryption for all messages in the underlying protocol, incurring a significant overhead.

3 Role assignment

In this section we present the ideal functionality $\mathcal{F}_{\mathsf{RA}}$[4], which assigns machines to computation roles while keeping this assignment hidden. (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.)

At a high-level, let us consider committee C consisting of c roles. There are two possible ways in which $\mathcal{F}_{\mathsf{RA}}$ chooses a machine for a role in C: (a) choosing a machine at random from among the set of honest machines (i.e. among the machines not corrupted so far), or (b) allowing the adversary to choose the machine, as long as the number of machines chosen by the adversary in C so far is within the allowed corruption bound (which is detemined as a function $\mathcal{T}$ on the fraction of corrupt machines). In the former case, $\mathcal{F}_{\mathsf{RA}}$ samples fresh keys, gives the (public) encryption and verification keys to everyone, and gives the corresponding (secret) decryption and signing keys only to the chosen machine. In the latter case, all keys are chosen by the adversary. The commands Nom-Honest and Nom-Corrupt capture the above kinds of nominations.

We need to ensure that the fraction of corruptions in a committee remains within the allowed bound until the nomination is completed. Looking ahead, to capture adaptive corruptions after the adversary has seen public keys generated via Nom-Honest but before Finish (which finalises the keys for a committee), we introduce an additional command Corrupt-Nominee.

Once a set of c machines are chosen for the committee C, $\mathcal{F}_{\mathsf{RA}}$ picks a random permutation on $[c]$ to determine which machine plays which role in C. Allowing $\mathcal{F}_{\mathsf{RA}}$ to map nominated machines to roles, instead of having machines assigned to specific roles in C a priori, prevents the adversary from targeting a specific role for corruption.

Further, there is a provision for each machine to

1. *'Read'*: this allows her to retrieve public keys corresponding to the roles that have been assigned, as well as to obtain secret keys if she has been assigned a role.

[4] Note this is not the same role assignment functionality as presented in [GHK+21]

2. *'Delete'*: this allows her to delete the secret keys corresponding to a role that she had been assigned. Looking ahead, this is used once she has finished preparing her one-time message but before she speaks.
3. *'Ready'*: this allows her to signal that she is available to be assigned a new role. This is typically used after she has finished executing her current role. We maintain both a global set of ready machines ("ready set"), as well as a committee-specific ready set. The latter keeps track of machines that have been ready throughout the nomination process for that committee.

If a machine that has been assigned a role gets corrupted after she has retrieved her secret keys (which she learns when she inputs 'read') but before she inputs 'delete', her secret keys are leaked to the adversary. However, if she gets corrupted after she inputs 'delete', her secret keys remain hidden. As we will see later, this is crucial for adaptive security.

The formal description of this ideal functionality $\mathcal{F}_{\mathsf{RA}}$ appears below. We assume $\mathcal{F}_{\mathsf{RA}}$ to be synchronous, with round switches occurring at the same time as the protocols using it.

We divide our role assignment functionality into two parts. The first describes the general setup and commands provided by parties for establishing new committees and reading generated keys. The second describes the powers allowed to the simulator, when populating committees under nomination with keys and the leakage in the case of corruption.

Functionality $\mathcal{F}_{\mathsf{RA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{D}, \mathtt{delay})$:

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.
- c: the 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.

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

New committee: After receiving $(\textsc{New}, \mathsf{cid}, C)$ from all honest parties in round r, store $(r, \mathsf{cid}, C, \mathsf{PKeys} = \emptyset, \mathsf{SKeys} = \emptyset, \mathtt{cor} = 0, \mathtt{nom} = 0, fin = \bot)$. 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, $\texttt{cor}$ and $\texttt{nom}$, 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 only proceeds to the next round if $fin = \top$ for all stored committees with round number $r' \leq r - \texttt{delay}$.

Read: On input $(\textsc{Read}, \mathsf{cid}, r)$ from M with identifier pid in round r, retrieve the value $(r^*, \mathsf{cid}^*, C, \mathsf{PKeys}, \mathsf{SKeys}, \texttt{cor}, \texttt{nom}, fin)$ where $\mathsf{cid} = \mathsf{cid}^*, r^* \leq r - \texttt{delay}$ 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 $fin = \bot$.
- Output $(\textsc{Delete}, \mathsf{pid})$ to $\mathcal{S}$.

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.

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.

Functionality $\mathcal{F}_{\mathsf{RA}}$ (continued):

Nominate honest: On input $(\textsc{Nom-Honest}, \mathsf{cid})$ from $\mathcal{S}$ in round r, retrieve the value $(r, \mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys}, \texttt{cor}, \texttt{nom}, fin)$. If no such value exists do nothing. If $\texttt{nom} < c$, do the following:
- Update $\texttt{nom} \leftarrow \texttt{nom} + 1$.
- 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 $\texttt{nom} = c$, go to procedure $\mathsf{Finish}(\mathsf{cid})$.

– Output (NOM-HONEST, cid, ek, vk) to $\mathcal{S}$.

Nominate corrupt: On input (NOM-CORRUPT, $\mathsf{cid}, \mathsf{pid}, (ek, vk), (dk, sk)$) from $\mathcal{S}$ in round r, retrieve the value $(r, \mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys}, \mathtt{cor}, \mathtt{nom}, fin)$. If no such value exists, do nothing. If $\mathtt{nom} < c$ and $\mathtt{cor} + 1 < \mathcal{T}(|\mathcal{I}|/|\mathcal{P}|)$, do the following:
– Update the nominated and corrupt counters $\mathtt{nom} \leftarrow \mathtt{nom} + 1, \mathtt{cor} \leftarrow \mathtt{cor} + 1$.
– Append $(ek, vk, \bot)$ to PKeys and $(\mathsf{pid}, dk, sk, \bot)$ to SKeys.
– If $\mathtt{nom} = c$, go to procedure $\mathsf{Finish}(\mathsf{cid})$.

Corrupt nominee: On input (CORRUPT-NOMINEE, $\mathsf{cid}, \mathsf{pid}$) from $\mathcal{S}$ in round r, retrieve the value $(r', \mathsf{cid}', C, \mathsf{PKeys}, \mathsf{SKeys}, \mathtt{cor}, \mathtt{nom}, fin)$ where $\mathsf{cid} = \mathsf{cid}'$ and $r = r'$. If no such value exists, do nothing. If $\mathtt{cor} + 1 < \mathcal{T}(|\mathcal{I}|/|\mathcal{P}|)$ and $\mathtt{cor} < \mathtt{nom}$, do the following:
– $\mathtt{cor} \leftarrow \mathtt{cor} + 1$
– Choose an element $(\mathsf{pid}', dk, sk, \bot)$ uniformly at random between the values of SKeys where $\mathsf{pid}' = \bot$.
– Update this value to be $(\mathsf{pid}, dk, sk, \bot)$
– Output (CORRUPT-NOMINEE, $\mathsf{cid}, \mathsf{pid}, dk, sk$) to $\mathcal{S}$.

Finish (cid): When the procedure $\mathsf{Finish}(\mathsf{cid})$ is called, retrieve the value $(r', \mathsf{cid}', C, \mathsf{PKeys}, \mathsf{SKeys}, \mathtt{cor}, \mathtt{nom}, fin)$ where $\mathsf{cid}' = \mathsf{cid}$ and do the following:
– Sample a random permutation ϕ on $[c]$.
– For the ith element of PKeys update $(ek, vk, \bot)$ to $(ek, vk, C_{\phi(i)})$.
– For the ith element of SKeys update $(\mathsf{pid}, dk, sk, \bot)$ as follows:
  • If $\mathsf{pid} = \bot$, 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, $\mathsf{cid}, \phi, \mathsf{PKeys}$) 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 $\mathcal{S}$.

4 Compiling abstract to natural YOSO

Consider an abstract YOSO-protocol in the $\mathcal{F}_{\mathsf{BC\&SPP}}$-hybrid model which is maliciously secure against a static adversary. This protocol is run by a set of committees, where each committee is associated with a set of roles. We may assume the execution of any honest role is completed by inputting at most one SEND command to an instance of $\mathcal{F}_{\mathsf{BC\&SPP}}$, this is enforced by the SPOKE token which kills the role.

The goal of our compiler is to transform such a statically-secure YOSO abstract protocol in the $\mathcal{F}_{\mathsf{BC\&SPP}}$-hybrid model into an adaptively-secure natural-world protocol in the $\mathcal{F}_{\mathsf{RA}}$-hybrid model, where $\mathcal{F}_{\mathsf{RA}}$ denotes the

ideal functionality for role assignment defined in Section 3. We also assume that the natural protocol has access to a bulletin board (formalized as an ideal functionality below) which can be used by anyone to broadcast a message.

Functionality $\mathcal{F}_{\mathsf{BB}}$

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

Overview of the compiler. Suppose we wish to compile an abstract protocol Π. At a high-level, the compiled protocol in the natural world involves the following stages: First, the machines initiate role assignment for committees that need to be nominated, which is determined based on the current round and the public state. Once the nomination process is completed, the machines can retrieve public keys corresponding to all roles in these committees and secret keys for the roles they were chosen for (if any). This can be done by machines inputting $\textsc{read}$ to $\mathcal{F}_{\mathsf{RA}}$.

Consider a machine M who has been assigned a role for some round of the abstract protocol. Recall that in this case, $\mathcal{F}_{\mathsf{RA}}$ provides M with a decryption key and a signing key. M obtains from $\mathcal{F}_{\mathsf{RA}}$ the signature verification keys of all the roles that are supposed to send messages to the role that's assigned to M, as well as the public encryption keys of the roles that its assigned role is supposed to send messages to. (Note that the latter key may not be available yet.) In this case M keeps asking $\mathcal{F}_{\mathsf{RA}}$ for these keys in each round. As soon as $\mathcal{F}_{\mathsf{RA}}$ provides these keys, the M is ready to execute the role R based on the specifications of the abstract protocol Π. Suppose this role R invokes $\mathcal{F}_{\mathsf{BC\&SPP}}$ in Π with a set of point-to-point and broadcast messages, then the machine does the following to emulate this step on behalf of the role:

- Read the bulletin board to retrieve messages posted by machines emulating sender roles. This includes broadcast messages and ciphertexts

encrypting point-to-point messages intended for R as a receiver, accompanied by signatures. Accept the messages only if the signatures are valid (note that the verification key of all roles are made public by $\mathcal{F}_{\mathsf{RA}}$).
- To retrieve the point-to-point message, uses the decryption key to decrypt the relevant ciphertexts.
- Proceed to compute the outgoing broadcast and point-to-point messages on behalf of the role R (Note that at this point, the machine has all the information a role holds in Π). Prepare a one-shot message comprising of the following **(a)** Broadcast messages **(b)** Ciphertexts encrypting the point-to-point messages using the encryption key of the relevant receiver roles in future committees (made public by $\mathcal{F}_{\mathsf{RA}}$) **(c)** Signature on these messages, computed using the signing key of R received from $\mathcal{F}_{\mathsf{RA}}$.
- Once the above one-shot message is computed, invoke $\mathcal{F}_{\mathsf{RA}}$ with input DELETE and delete its own entire state, except the one-shot message to be posted. In particular, delete the secret keys, received messages and randomness used on behalf of the role R.
- Post this message to the bulletin board (as an atomic action).

Once the machine M has finished executing the role R, it notifies $\mathcal{F}_{\mathsf{RA}}$ that she is READY i.e. available to be assigned a new role.

We point out that in the above informal description, we focused on machines that were assigned computation roles. The compiler easily accommodates actions by input and output roles in Π as well – the only difference is that these roles are carried out by fixed machines and their identity is not secret. Therefore, the public keys of these roles can be established via a PKI and need not be handled by $\mathcal{F}_{\mathsf{RA}}$. Further, the messages posted on the bulletin board by machines executing these roles need not be signed.

Protocol Compile(Π)

Notation: The algorithm $\mathsf{Nominate}(r, \{\mathsf{Broadcast}_{\mathsf{sid}}\}_{\mathsf{sid}\in\mathsf{SID}})$ denotes a publicly computable function which when given the current round and public state outputs the set of committees to be nominated.

Init: Initialise sets of messages and keys for each role:
- For each $\mathsf{R} \in \mathsf{Role}$ and $\mathsf{sid} \in \mathsf{SID}$ define a set $\mathsf{R}.\mathsf{Rec}_{\mathsf{sid}} \leftarrow \emptyset$ of ciphertexts sent to the role. $\mathsf{R}.ek \leftarrow \perp$, $\mathsf{R}.vk \leftarrow \perp$, $\mathsf{R}.dk \leftarrow \perp$ and $\mathsf{R}.sk \leftarrow \perp$.
- If $\mathsf{R} \in \mathsf{Role}^{\text{IN}} \cup \mathsf{Role}^{\text{OUT}}$, set $\mathsf{R}.ek$ and $\mathsf{R}.vk$ to relevant public keys established by PKI.
- For each $\mathsf{sid} \in \mathsf{SID}$: $\mathsf{Broadcast}_{\mathsf{sid}} = \emptyset$.

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

Role Keys: Once the machine finishes nominating committees in a round r, it proceeds to read the keys for the committees nominated in the previous round. For each committee, the machine inputs $(\textsc{Read}, \mathsf{cid}, r)$ to $\mathcal{F}_{\mathsf{RA}}$ receiving lists PKeys and SKeys.

- For each element (ek, vk, R') in PKeys the machine stores the role keys as $\mathsf{R}'.ek \leftarrow ek$ and $\mathsf{R}'.vk \leftarrow vk$.
- For each element $(\mathsf{pid}, dk, sk, \mathsf{R})$ in SKeys (where pid corresponds to the machine's identifier) store the keys $\mathsf{R}.dk \leftarrow dk, \mathsf{R}.sk \leftarrow sk$. *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 $(\textsc{Read}, \mathsf{sid}, r-1)$ to $\mathcal{F}_{\mathsf{BB}}$, for each output element (M', r', msg') it receives the machine does the following:

- Parse msg' as $((\mathsf{S}, \mathsf{sid}, (\mathsf{R}_1, \overline{x}_1), \ldots, (\mathsf{R}_k, \overline{x}_k), x), \sigma)$
- Verifies the signature $b \leftarrow \mathsf{SIG.Verify}(\mathsf{S}.vk, (\mathsf{S}, \mathsf{sid}, (\mathsf{R}_1, \overline{x}_1), \ldots, (\mathsf{R}_k, \overline{x}_k), x), \sigma)$, ignoring the message if verification does not succeed [a].
- Add (S, x) to $\mathsf{Broadcast}_{\mathsf{sid}}$.
- For $i \in [k]$ add $(\mathsf{S}, \overline{x}_i)$ to $\mathsf{R}_i.\mathsf{Rec}_{\mathsf{sid}}$.

If any role has more than one message with a valid signature, both should be ignored.

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

Send $\mathcal{F}_{\mathsf{BC\&SPP}}$: When the role $\mathsf{R} \in \mathsf{Role}^{\textsc{In}} \cup \mathsf{Role}^{\textsc{Comp}}$ assigned to M outputs $(\textsc{Send}, \mathsf{R}, ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k)), x)$ to $\mathcal{F}_{\mathsf{BC\&SPP}}$ with session identifier sid do the following:

1. For $j \in [k]$: $\overline{x}_j \leftarrow \mathsf{PKE.Enc}^{(\mathsf{sid},\mathsf{R})}(\mathsf{R}_j.ek, x_j; \rho_j)$.
2. Let $\mathsf{msg} = (\mathsf{R}, r, \mathsf{sid}, (\mathsf{R}_1, \overline{x}_1), \ldots, (\mathsf{R}_k, \overline{x}_k), x)$.
3. Compute $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{R}.sk, \mathsf{msg})$ and set $\mathsf{msg}' = (\mathsf{msg}, \sigma)$ [b].
4. If $\mathsf{R} \in \mathsf{Role}^{\textsc{Comp}}$
   - Input $(\textsc{Delete})$ to $\mathcal{F}_{\mathsf{RA}}$.

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

5. Post msg' to the bulletin board.
6. Input (READY) to $\mathcal{F}_{\mathsf{RA}}$ if $\mathsf{R} \in \mathsf{Role}^{\text{COMP}}$.

If a machine has been assigned multiple roles it should activate them until they have all sent a message or completed the round, collecting all their messages at Step 6.2 and posting them together.

[a] this verification is not needed if $\mathsf{S} \in \mathsf{Role}^{\text{IN}} \cup \mathsf{Role}^{\text{OUT}}$

[b] Here, signatures can be avoided if $\mathsf{R} \in \mathsf{Role}^{\text{IN}}$.

5 Security of the compiler

In this section, we prove the security of the compiler presented in Section 4 which transforms a *static, abstract* YOSO protocol to an *adaptively-secure* natural protocol. The security of our compiled *natural* protocol fundamentally relies on the security of the original *abstract* protocol. The primary challenge arises due to the difference in the adversary's corruption powers between the abstract and natural world. In order to rely on the static security of abstract protocol, we must be able to translate the adaptive adversary in the natural world to an appropriate static adversary in the abstract world (against which a simulator must exist, due to security of the abstract protocol).

Theorem 2. *Consider an abstract protocol Π which YOSO securely implements the ideal functionality $\mathcal{F}$ in the presence of c/w static corruptions (where $c = \Omega(\kappa)$ 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 $\mathcal{F}_{\mathsf{RA}}$ be shorthand for $\mathcal{F}_{\mathsf{RA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{U}, 2)$ where $\mathcal{U}$ samples the uniform distribution and $\mathcal{T}(f) = c\left(1 - (1-\epsilon)(1-f)^2\right)$, for $\epsilon > 0$. Suppose Π is secure against an arbitrary number of static corruptions in the input and output roles and has the following properties [5]:*

- *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.*

[5] Note that all existing *abstract* YOSO protocols (such as the protocols in [GHK+21], [KRY22]) satisfy these properties.

Then the protocol $\mathsf{Compile}(\Pi)$ *UC implements the ideal functionality* $\mathcal{F}$ *in the* $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{RA}})$*-hybrid model, under the presence of* $T < Nf_t$ *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* $0 < 1 - 2wf_t + wf_t^2$*.* [6]

Proof. We begin with the description of our simulator $\mathcal{S}_{\mathtt{nat}}$ for the compiled protocol in the natural world.

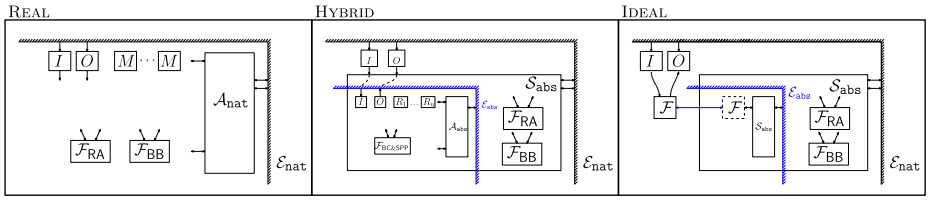

Fig. 3: Our high level proof strategy for exploiting the abstract simulator

At a very high-level, $\mathcal{S}_{\mathtt{nat}}$ chooses a random set of roles to corrupt statically, with respect to which $\mathcal{S}_{\mathtt{abs}}$ (the simulator of underlying abstract protocol Π) is run. Towards the adaptive adversary in the natural world (which is in the $\mathcal{F}_{\mathsf{RA}}$ hybrid), $\mathcal{S}_{\mathtt{nat}}$ emulates this functionality of $\mathcal{F}_{\mathsf{RA}}$ with the following major difference: Instead of choosing the roles corresponding to corrupt nominations at random, these are appropriately matched to the above pre-determined set of static corrupt roles. Further, $\mathcal{S}_{\mathtt{nat}}$ acts as a proxy between the adversary $\mathcal{A}_{\mathtt{nat}}$ in the natural world and $\mathcal{S}_{\mathtt{abs}}$. More specifically, the messages sent by $\mathcal{S}_{\mathtt{abs}}$ on behalf of honest roles are encrypted with suitable encryption keys (determined as part of $\mathcal{F}_{\mathsf{RA}}$) and posted on the bulletin board in the natural world. Similarly, the messages posted by $\mathcal{A}_{\mathtt{nat}}$ in the natural world are decrypted to emulate corrupt roles towards $\mathcal{S}_{\mathtt{abs}}$. This completes a high-level description of the simulator.

Simulator $\mathcal{S}_{\mathtt{nat}}$

We define a simulator $\mathcal{S}_{\mathtt{nat}}$ which proceeds as follows:

– The simulator chooses a random subset $\mathcal{I}_C$ of the roles in each committee C to corrupt, where $|\mathcal{I}_C| = \tau$. Here $\tau < c/2$. Let $\mathcal{H}_C = C \setminus \mathcal{I}_C$.

[6] For $w = 2$ this allows $f_t \leq (1 - \sqrt{1/2}) \approx 0.29$

– The environment has the ability to corrupt and activate machines. For corruptions the adversary expects to see a role with some probability. For activations, it expects to see messages posted to the bulletin board, if machine was assigned a role which has produced output. To account for these cases the simulator then defines a list $\mathcal{R}_C = \sigma(\mathcal{I}_C)||\sigma'(\mathcal{H}_C)$ where σ and σ' are random permutations on the sets of corrupt and honest roles respectively. The roles we revealed to the adversary as corruptions will be taken from the front of this list, while roles which finish are taken from the back. This ensures the roles in our corruption budget are not forced to finish before the roles which must remain honest.

The simulator runs the adversary $\mathcal{A}_{\texttt{nat}}$. When $\mathcal{A}_{\texttt{nat}}$ invokes the ideal functionality $\mathcal{F}_{\mathsf{RA}}$, the simulator runs the following modified version of $\mathcal{F}_{\mathsf{RA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{U}, 2)$ which it presents to the adversary:

– When the committee is finished being nominated, remove the first `cor` elements of $\mathcal{R}_C$, assigning them to the corrupt nominations.
– The functionality does not fix the mapping from roles to machines, rather for every adaptive corruption after the nomination of the committee is finished, over $|\mathcal{R}_C|$ trials remove the role at the front of $\mathcal{R}_C$ and assign to the corrupted machine with probability $1/|\mathcal{H}|$. Any role assigned to the newly corrupted machine should be activated, so it reaches the state it would be in, given the number of activations to the newly corrupted machine in the current round. Once t elements have been removed from the front of $\mathcal{R}_C$, never assign any new role to an adaptively corrupted machine.
– When machines finish their computation for a round they have a chance to be assigned a role which they will output for. On the last activation before they are ready to proceed to the next round. The environment fixes the order in which machines are activated in a round. For the ith honest machine which the environment activates, in the round in which committee C is ready to speak, over $|\mathcal{R}_C|$ trials, with probability $1/(|\mathcal{H}| - i + 1)$, remove the element R at the back of $\mathcal{R}_C$ and post the output of R to the bulletin board on behalf of M. Removing the role from the list corresponds to $\mathcal{F}_{\mathsf{RA}}$ assigning the role to the machine. If M was not ready at any point during the nomination of this role it should not post the message.

If the adversary ever provides a valid signature for an honest role, which differs from the signature produced during output from that role, simply ignore the associated message.
The simulator $\mathcal{S}_{\mathtt{nat}}$ computes messages for the honest parties by running the abstract simulator $\mathcal{S}_{\mathtt{abs}}$ and acting as its environment. In particular, $\mathcal{S}_{\mathtt{abs}}$ is the simulator for the adversary which simply forwards all messages to and from the environment.
In the beginning of the abstract simulation for each committee C, our simulator statically corrupts the parties in $\mathcal{I}_C$. The simulator $\mathcal{S}_{\mathtt{nat}}$ interacts with $\mathcal{S}_{\mathtt{abs}}$ by:

- When $\mathcal{S}_{\mathtt{abs}}$ passes on messages from $\mathcal{F}_{\mathsf{BC\&SPP}}^{\mathsf{sid}}$, $\mathcal{S}_{\mathtt{nat}}$ encrypts each point-to-point message to its intended recipient, using the keys from $\mathcal{F}_{\mathsf{RA}}$. For messages between honest roles, $\mathcal{S}_{\mathtt{abs}}$ will only output their length ℓ. In this case $\mathcal{S}_{\mathtt{nat}}$ encrypts $\mathbf{0}^{\ell}$. These encryptions are then posted on the bulletin board.
- When the adversary posts a message on $\mathcal{F}_{\mathsf{BB}}$ from a corrupt role, the simulator decrypts all messages to honest parties and passes the plaintexts on to $\mathcal{S}_{\mathtt{abs}}$ to post on $\mathcal{F}_{\mathsf{BC\&SPP}}^{\mathsf{sid}}$.
- When $\mathcal{S}_{\mathtt{abs}}$ invokes its ideal functionality, forward the messages to the ideal functionality $\mathcal{F}$ in the natural world to obtain the response, which can be subsequently forwarded to $\mathcal{S}_{\mathtt{abs}}$.

Next, we proceed through a series of hybrids which will enable us to construct our desired abstract adversary, for which we know there exists a simulator. This will allow us to rely on the security of the abstract protocol which we are compiling.

Hybrid H_0*:* Run the real experiment $REAL_{\Pi_{\mathtt{nat}},\mathcal{A}_{\mathtt{nat}},\mathcal{E}_{\mathtt{nat}}}(1^{\kappa})$

Hybrid H_1*:* In this series of hybrids we will reprogram $\mathcal{F}_{\mathsf{RA}}$, so the simulator may sample which roles will be revealed at the beginning. We reprogram $\mathcal{F}_{\mathsf{RA}}$ to behave as described in the simulator.

Hybrid H_2*:* Same as previous, except ignore the message on $\mathcal{F}_{\mathsf{BB}}$ if the adversary ever provides a valid signature for an honest role, which differs from the signature produced during output from that role. We may now be sure that the messages received from honest roles do in fact correspond to their real output.

Hybrid H_3*:* We now wish to move the roles in $\mathcal{H}_C$ which the simulator is running using virtual versions of $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$ to be honest roles in the abstract world.

We observe that the honest roles in $\mathcal{H}_C$ all have consistent views of the messages in their virtual versions $\mathcal{F}_{\mathsf{BC\&SPP}}$ due to the properties of $\mathcal{F}_{\mathsf{BB}}$, and the guarantee that messages from honest roles seen in $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$ actually come from those honest machines. Therefore, the honest roles will exhibit the same behaviour if allowed to interact through an actual $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$.

We let our simulator act as the environment toward an abstract adversary $\mathcal{A}_{\mathtt{abs}}$ of our design. The simulator specifies the static corruptions in each committee to be $\mathcal{I}_C$ as sampled previously. Our abstract adversary $\mathcal{A}_{\mathtt{abs}}$ simply takes the messages provided to it by its environment and sends them on behalf of the corrupted roles. Whenever a message is encrypted to an honest role and posted on $\mathcal{F}_{\mathsf{BB}}$ the simulator decrypts this message and provides it to $\mathcal{A}_{\mathtt{abs}}$. When $\mathcal{A}_{\mathtt{abs}}$ receives information from $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$ it relays this to the simulator. When the natural adversary reads from $\mathcal{F}_{\mathsf{BB}}$, the simulator in turn requests that $\mathcal{A}_{\mathtt{abs}}$ read all broadcast messages from the last round. The simulator may then encrypt the real messages for corrupt receivers, and use the leaked message length ℓ to encrypt $\mathbf{0}^\ell$ for messages between honest roles.

Hybrid H_4*:* In the previous hybrid we had defined an abstract real world adversary $\mathcal{A}_{\mathtt{abs}}$ used to produce messages for each of the honest roles. Now we replace said adversary, by its ideal world counterpart $\mathcal{S}_{\mathtt{abs}}$. This is possible as the ideal functionalities in the natural and abstract worlds permit the same leakage, allowing the simulator to forward any messages between $\mathcal{S}_{\mathtt{abs}}$ and $\mathcal{F}$. The simulator no longer needs any information on the honest inputs beyond what is provided by $\mathcal{F}$, and is therefore in the ideal world.

We briefly summarise the fraction of allowable corruptions through our hybrids. During the nomination of a committee up to $\mathcal{T}(f)$ of the roles were allowed to be corrupted. After nomination was finished, we at most allow $\tau - \mathcal{T}(f)$, further adaptive corruptions. We require $\tau > (1+\epsilon)cf^* + \mathcal{T}(f)$, where f is the fraction of corruptions when nominations finished, and $f^* = f_t - f$ is the remaining corruption budget at that point. This accounts for any adaptive corruptions caused by the adversary expending f^* corruption budget after nomination was complete For $\mathcal{T}(f) = c\left(1-(1-\epsilon')(1-f)^2\right)$ this gives,

$$\tau > c(1 + (1+\epsilon)f^* - (1-\epsilon')(1-f)^2).$$

We note that as $N = R_{max}^{2+\delta}$ for $\delta \geq 1$, if R_{max} is at least κ roles the probability of a collision will be negligible. This leaves us with the requirement,

$$c/w > c(1 + (1 + \epsilon)f^* - (1 - \epsilon')(1 - f)^2),$$

to achieve our desired honest fraction. By a similar analysis to that of Benhamouda *et al.* [BGG+20] the above equation is possible to satisfy when $0 < 1 - wf_t - wf + wf^2$, where the right-hand side is minimized when $f = f_t$. Therefore, we only tolerate a corruption threshold $0 < 1 - 2wf_t + wf^2$, which for $w = 2$ allows $f_t < 1 - \sqrt{1/2} \approx 0.29$.

To complete our proof we need to argue the indistinguishability of each pair of hybrids. We note that the most challenging aspect of the proof is to show that reprogramming the $\mathcal{F}_{\mathsf{RA}}$ maintains indistinguishability. Due to the involved nature of this argument, we first discuss this in a separate section below and then move on to argue the indistinguishability of each pair of hybrids.

5.1 Re-programming role assignment

We wish to model how the role assignment mechanism distributes roles among machines. We then model the adversaries ability to corrupt machines seeing assigned roles as well as its control of the activation of honest machines. Our final goal is to ensure that we always have sufficient corruption budget regardless of how the adversary interleaves corruptions and activations.

Let us consider the following game played by an unbounded adversary.

Game. The challenger places R balls, labeled $1, \ldots, R$, uniformly at random into H bins. The adversary is then allowed to check up to T of these bins one-by-one and *corrupting* the labels of balls in the opened bin in a random order. The adversary may additionally remove an unlimited number of bins from the game, receiving any labels they contain, we consider these labels to be *removed* and not *corrupted* by the adversary. We may describe the assignment of the balls to bins by random variables $B_1, \ldots B_R$, which are i.i.d. uniformly across $[H]$.

Additionally, let $L_1, \ldots, L_R$ be the random variables where each L_i takes the value of the ith label corrupted by the adversary, with distributions taken over the random coins of the challenger and adversary. If the adversary never finds at least i labels, we let $L_i = 0$. We make the following observations:

– Consider an adversary which has checked or removed j bins, the indices of which are in the set C. For any choice of $(j+1)$th bin the probability the adversary will see ball i which it has not yet corrupted or removed is

$$\Pr[B_i = k | \forall \ell \in C : B_i \neq \ell \wedge k \neq \ell, |C| = j] = \frac{1}{H - j}.$$

– No label is observed twice, $\Pr[L_i = L_j | i \neq j, L_i \neq 0] = 0$, and the next label corrupted by the adversary is always uniform across the uncorrupted and not removed labels,

$$\Pr[L_i = k | k \neq 0 \wedge \forall j \in [i-1] : L_j \neq k] = \frac{1}{R - i}.$$

Following our observations we will make a number of modifications to the game, which are indistinguishable even for a computationally unbounded adversary.

Decouple. The first modification we make to this game is placing balls into bins lazily. For a ball labeled $i \in [R]$, rather than placing the ball into any bin at the beginning of the game we may instead add the ball to the $(j+1)$th opened or removed bin with probability $\frac{1}{H-j}$, until the ball is found. This produces an identical distribution to that previously observed by the adversary. By independence of the balls, the same may be extended to every ball.

Relabel. For our second modification we wish to sample the order in which labels are revealed to the adversary at the beginning of the game. We still sample the ball placement lazily as previously, simply relabelling the revealed balls to match a random permutation $\ell_1, \ldots \ell_R$ of the elements in $[R]$ which is picked at the beginning of the game. Labels which are corrupted by the adversary are taken from the beginning of this list, while labels which are removed are taken from the end. As the next corrupted or removed by the adversary is always uniform among the remaining labels, sampling this beforehand will lead to the same distribution; the list is always a uniformly random permutation on the remaining elements.

Bound. For our final modification the after the adversary corrupts B labels the challenger returns the empty set for any additional bins the adversary opens. Note, balls are still added to removed buckets as previously.

This game is statistically indistinguishable from the previous game, if $R = \Omega(\kappa)$ and $\frac{B}{R} = (1+\epsilon)\frac{T}{H}$ for positive ϵ and sufficiently large H. First we restrict our adversary to one which chooses all bins it will open in the beginning of the game.

Consider the scenario where balls are placed into bins with replacement, i.e. each ball is independently added to each bin with probability $1/H$. In this case the contents of the bins are entirely independent of one another. By a union bound, the probability that such an adversary corrupt any balls in a given opened bin is at most R/H. We may then bound the number of non-empty bins this adversary corrupts by considering $S \sim Bin(T, \frac{R}{H})$. By a Chernoff bound,

$$\Pr\left[S > (1+\epsilon_1)\frac{TR}{H}\right] < \exp\left(-\frac{R(T/H)\epsilon_1^2}{2+\epsilon_1}\right),$$

which is negligible for $R = \Omega(\kappa)$. In the above scenario the adversary clearly gains no additional advantage by choosing bins adaptively, as an opened bin provides no information on the unopened bins.

Restricting balls to only be placed into a single bin strictly reduces the probabilities that any given bin, at any given point, is non-empty, causing the same bound to apply.

To bound the number of balls the adversary sees we must additionally bound the number of collisions among the balls. For this we consider $C \sim Bin(\frac{R(R-1)}{2}, \frac{1}{H})$. Let $H = R^{2+c}$ for $c > 0$, note the number of collisions will be a less than one in expectation, can be tail bounded as a small constant k. Let $\epsilon_2 = kR^c - 1$, then

$$\begin{aligned}\Pr[C > k] &< \Pr[C > (1+\epsilon_2)R^{-c}] < \Pr\left[C > (1+\epsilon_2)\frac{R(R-1)}{2H}\right] \\ &< \exp\left(-\frac{(R(R-1)/(2H))\epsilon_2^2}{2+\epsilon_2}\right),\end{aligned}$$

which is negligible for $k = \Omega(1)$ and $R = \Omega(\kappa)$. The total number of balls corrupted may then be bounded by $B = (1+\epsilon_1)\frac{TR}{H} + k$, where $\frac{B}{R} = (1+\epsilon')\frac{T}{H}$ for $\epsilon' > 0$.

The final game. In summary, we have arrived at a new game which is indistinguishable from the original. There are R balls and H bins, let $B = (1+\epsilon)\frac{TR}{H}$ for $\epsilon > 0$. In this game the challenger randomly permutes all the labels to obtain $\ell_1, \ldots, \ell_B, \ldots, \ell_R$.

The adversary may then open bins one-by-one. When the adversary opens or removes the jth bin, for each ball which has not been corrupted

or removed the challenger will add a ball to the bin with probability $1/(H - j + 1)$. When the ball is added it is given the next label in the sequence sampled at the beginning of the game, with labels for corrupted balls being taken from the beginning of the list and removed balls the end. Once B balls have been corrupted, the adversary is always shown empty for every bin it opens up to the threshold T. When bins are removed each remaining ball is added to the bin with probability $1/(H - j + 1)$, with labels taken from the back of the list.

5.2 Indistinguishability of hybrids

$\mathsf{H}_0 \approx \mathsf{H}_1$*:* When the assignment of committee C is finished, i.e. the functionality executes (FINISH, cid), at most $\mathcal{T}(f)$ of the roles in the committee have been assigned to corrupt parties. As the roles given to corrupt machines are defined by a random permutation this will be indistinguishable to receiving the first $\mathcal{T}(f)$ roles of $\mathcal{R}_C$. For the remainder of the roles in $\mathcal{R}_C$ we must show that fixing the roles which we will allow to be corrupted, and assigning them to machines on the fly, is indistinguishable to the adversary.

Recall the first game described in Section 5.1, this corresponds to the case where the mapping from roles to machines is fixed statically. For our purposes the labels correspond to identifiers for the remaining roles in $\mathcal{R}_C$, while the honest machines correspond to the bins. When the adversary corrupts a machine it sees the state associated with corrupting the bucket with the same index. If a machine is allowed to finish executing a role, producing output, this is equivalent to removing the bucket of the same index.

The final game in Section 5.1, in turn corresponds to the setting where the roles mapped to a machine are chosen at the time of corruption, or final activation in a round. As these two games are statistically indistinguishable for an unbounded adversary, changing the behaviour of $\mathcal{F}_{\mathsf{RA}}$ according to the challenger in the final game will also be indistinguishable.

$\mathsf{H}_1 \approx \mathsf{H}_2$*:* Due to the previous hybrids we may now be sure that any role in $\mathcal{H}_C$ will never be revealed by the adaptive corruption of a machine. Therefore, for any honest role $\mathsf{R} \in \mathcal{H}_C$, we may reduce an adversary causing the simulator to abort to one winning the unforgeability game of the signature scheme with the same advantage. This leads to a series of hybrids, with $|\mathcal{H}_C|$ hybrids for each committee C.

$\mathsf{H}_2 \approx \mathsf{H}_3$*:* We wish to replace ciphertexts between honest roles through a reduction to the security properties of our encryption scheme. However, to provide the necessary inputs to provide the necessary inputs to $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$ we must maintain the ability to decrypt messages sent to honest parties. For each pair of honest roles $\mathsf{S} \in \mathcal{H}_C$ and $\mathsf{R} \in \mathcal{H}_{C'}$ we may replace any encryption of x_{R} between S and R for a session identifier sid with an encryption of $\mathbf{0}^{|x_{\mathsf{R}}|}$. An adversary distinguishing these cases may be reduced to an adversary winning the labeled CCA game of the encryption scheme with the same advantage. Messages sent to R from other senders may still be decrypted and input to $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$ as they have different labels.

By the properties of $\mathcal{F}_{\mathsf{BB}}$ the views each honest has of $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$ will be consistent. Running the honest roles in an abstract protocol will produce the same messages. Note, we do not receive messages between two honest parties, however $\mathcal{F}^{\mathsf{sid}}_{\mathsf{BC\&SPP}}$ will leak their lengths ℓ, allowing encrypting $\mathbf{0}^{\ell}$ as before.

$\mathsf{H}_3 \approx \mathsf{H}_4$*:* By the security of the abstract protocol, for any real world adversary $\mathcal{A}_{\mathtt{abs}}$ there exists an ideal world adversary $\mathcal{S}_{\mathtt{abs}}$ such that for all environments $\mathcal{E}_{\mathtt{abs}}$,

$$REAL_{\Pi_{\mathtt{abs}},\mathcal{A}_{\mathtt{abs}},\mathcal{E}_{\mathtt{abs}}}(1^{\kappa}) \approx IDEAL_{\mathcal{F},\mathcal{S}_{\mathtt{abs}},\mathcal{E}_{\mathtt{abs}}}(1^{\kappa}).$$

The environment we consider is a composition of the remainder of natural simulator and the natural environment, which we call $\mathcal{E}_{\mathtt{abs}}$. The efficiency of this environment is preserved under composition. The natural simulator appropriately specifies static corruptions at the beginning of the experiment, making it a permissible environment for the abstract world. As the ideal functionalities of the two settings are identical, replacing $\mathcal{A}_{\mathtt{abs}}$ by the simulator $\mathcal{S}_{\mathtt{abs}}$ allowed to interact with $\mathcal{F}$ will be indistinguishable to $\mathcal{E}_{\mathtt{abs}}$ and therefore also $\mathcal{E}_{\mathtt{nat}}$.

6 Compiling computationally secure protocols

Our compiler in Section 4 supports the class of YOSO protocols in the $\mathcal{F}_{\mathsf{BC\&SPP}}$-hybrid model, such as the information-theoretic protocol of [GHK+21]. However, this notably excludes protocols which assume explicit access to keys for the roles to allow zero-knowledge proofs or any other types of public verifiability for point-to-point messages. A large part of the existing YOSO protocol literature falls under this umbrella, including the protocols presented in [BDO22,KRY22] and the computationally secure protocol of [GHK+21].

Kolby et al. [KRY22] introduced the verifiable state propagation (VeSPa) functionality $\mathcal{F}_{\mathsf{VeSPa}}$ to capture verifiability of point-to-point messages and designed protocols in the $(\mathcal{F}_{\mathsf{VeSPa}}, \mathcal{F}_{\mathsf{BC\&SPP}})$-hybrid model instead. We show how our compiler may be extended to accommodate the compilation of protocols in the $(\mathcal{F}_{\mathsf{VeSPa}}, \mathcal{F}_{\mathsf{BC\&SPP}})$-hybrid model. This extended compiler would captures the verification requirements of a broad range of existing protocols.

6.1 Verifiable state propagation

In this section, we recap the verifiable state propagation (VeSPa) functionality $\mathcal{F}_{\mathsf{VeSPa}}$ introduced in Kolby et al. [KRY22]. Informally, this functionality enables both point-to-point and broadcast communication, while allowing the sender to prove that she correctly computed these messages (based on messages she received and possibly other additional inputs).

Functionality $\mathcal{F}_{\mathsf{VeSPa}}$

This ideal functionality has the following behaviour:

- Define a map $\mathcal{R} : \mathsf{Role} \to \mathsf{Rel}_\perp$. *Specify the relations the messages of each role must satisfy.*
- Initially create point-to-point and broadcast maps:
  $y : \mathbb{N} \times \mathsf{Role} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ where $y(r, \mathsf{R}, \mathsf{R}') = \perp$ for all $r, \mathsf{R}, \mathsf{R}'$
  $m : \mathbb{N} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ where $m(r, \mathsf{R}) = \perp$ for all r, R.
- On input $(\textsc{Send}, \mathsf{S}, ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k)), x, w)$ in round r proceed as follows:
  - Let $\phi_{send} = ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k))$ and $\phi_{broadcast} = x$.
  - Let ϕ_{public} be the current public state, represented by a vector of all elements $(r, \mathsf{R}, \mathsf{msg})$, where $m(r, \mathsf{R}) = \mathsf{msg} \neq \perp$.
  - Collect all $y_k \neq \perp$ for $r' < r, \mathsf{R}' \in \mathsf{Role}$ where $y(r', \mathsf{R}', \mathsf{S}) = y_k$ to produce a vector $\phi_{receive} = ((\mathsf{R}'_1, y_1), \ldots, (\mathsf{R}'_m, y_m))$.
  - If $((\phi_{send} || \phi_{receive} || \phi_{broadcast} || \phi_{public}), w) \notin \mathcal{R}(\mathsf{S})$ ignore the input.
  - Else:
    - For $i \in [n]$ update $y(r, \mathsf{S}, \mathsf{R}_i) = x_i$. *Store point to point messages from the role.*
    - Update $m(r, \mathsf{S}) = x$. *Store the broadcast message from the role.*
    - Output $(\mathsf{S}, ((\mathsf{R}_1, |x_1|), \ldots, (\mathsf{R}_k, |x_k|)), x)$ to $\mathcal{S}$. For corrupt roles R_i output x_i to $\mathcal{S}$. *Leak messages lengths and the broadcast message to the simulator in a rushing fashion.*

If S is honest give SPOKE to S.
- On input $(\text{READ}, \mathsf{R}, \mathsf{S}, r')$ in round r where $r' < r$ for $x = y(r', \mathsf{S}, \mathsf{R})$ output x to R.
- On input $(\text{READ}, \mathsf{S}, r')$ in round r where $r' < r$ output $x = m(r', \mathsf{S})$ to R.

6.2 Extending to verifiable state propagation

Our compiler lends itself well to extension to the $\mathcal{F}_{\mathsf{VeSPa}}$-hybrid model, as we have made relatively few requirements of the encryption scheme used for underlying protocol messages. If the encryption scheme used additionally permits efficient proofs of knowledge of plaintext, we may use non-interactive zero-knowledge to prove that the encrypted messages between roles satisfy the relations required by $\mathcal{F}_{\mathsf{VeSPa}}$. In our extension of the compiler we use the NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ introduced by [GOS12]. Looking ahead, the ability to extract witnesses through $\mathcal{F}_{\mathsf{VeSPa}}$ means that we no longer require CCA security for our encryption scheme and may relax this to CPA security. We define a relation $\mathcal{R}_{\mathsf{VeSPa}}$ which describes what we require of the messages sent by our machines. The requirements may be divided into two categories:

- Encryption and decryptions is performed correctly.
- The incoming and outgoing plaintexts, and the public state satisfy the relation required by $\mathcal{F}_{\mathsf{VeSPa}}$ in the underlying protocol.

For a message $\mathsf{msg} = (\mathsf{R}, \mathsf{sid}, (\mathsf{R}_1, \overline{x}_1), \ldots, (\mathsf{R}_k, \overline{x}_k), x)$, incoming message set $\mathsf{R.Rec_{sid}}$, with elements of the form $(\mathsf{S}, \overline{x}_i)$, and past broadcast messages $\mathsf{Broadcast_{sid}}$, with elements of the form (R, x), we define our relation,

$$\mathcal{R}_{\mathsf{VeSPa}} = \left\{ \begin{array}{l|l} \phi = \begin{pmatrix} \mathsf{R}, \mathsf{sid}, \mathsf{R}.ek, \\ \mathcal{R}_{\mathsf{sid}}(\mathsf{R}), \\ (\mathsf{R}_j.ek)_{j\in[k]}, \\ \mathsf{R.Rec_{sid}}, \\ \mathsf{msg}, \\ \mathsf{Broadcast_{sid}} \end{pmatrix} & \top = \mathsf{KeyMatch}(\mathsf{R}.dk, \mathsf{R}.ek)^7 \\ & \text{For } j \in [k]: \\ & \quad \overline{x}_j = \mathsf{PKE.Enc}(\mathsf{R}_j.ek, x_j; \rho_j) \\ & \text{For } (\mathsf{S}, \overline{y}_j) \in \mathsf{R.Rec_{sid}}: \\ & \quad y_j = \mathsf{PKE.Dec}(\mathsf{R}.dk, \overline{y}_j) \\ & \phi_{send} = ((\mathsf{R}_j, x_j))_{j\in[k]} \\ & \phi_{rec} = ((\mathsf{R}_j, y_j))_{(\mathsf{S},\overline{y}_i)\in\mathsf{R.Rec_{sid}}} \\ w = \begin{pmatrix} \mathsf{R}.dk, \\ (x_j, \rho_j)_{j\in[k]}, \\ w' \end{pmatrix} & \phi_{bc} = x \\ & \phi_{pub} = \mathsf{Broadcast_{sid}} \\ & ((\phi_{send}, \phi_{rec}, \phi_{bc}, \phi_{pub}), w') \in \mathcal{R}_{\mathsf{sid}}(\mathsf{R}) \end{array} \right\}.$$

The only changes we need to allow for this functionality are when dealing with messages sent via $\mathcal{F}_{\mathsf{VeSPa}}$, the role assignment process remains unchanged.

Protocol Extended Compile(Π)

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 $(\textsc{Read}, \mathsf{sid}, r-1)$ to $\mathcal{F}_{\mathsf{BB}}$, for each output element (M', r', msg) it receives the machine does the following:

- Parse msg as $((\mathsf{S}, \mathsf{sid}', (\mathsf{R}_1, \overline{x}_1), \ldots, (\mathsf{R}_k, \overline{x}_k), x, \pi), \sigma)$
- If sid' is the session identifier for an instance of $\mathcal{F}_{\mathsf{VeSPa}}$ proceed with these steps, otherwise handle the message as done for $\mathcal{F}_{\mathsf{BC\&SPP}}$ in the original compiler.
- Verifies the signature $b \leftarrow \mathsf{SIG.Verify}(\mathsf{S}.vk, (\mathsf{S}, (\mathsf{S}, \mathsf{sid}', (\mathsf{R}_1, \overline{x}_1), \ldots, (\mathsf{R}_k, \overline{x}_k), x), \pi), \sigma)$, ignoring the message if verification does not succeed.
- Defines the statement $\phi \leftarrow (\mathsf{R}, \mathsf{sid}', \mathsf{R}.ek, \mathcal{R}_{\mathsf{sid}'}(\mathsf{R}), (\mathsf{R}_j.ek)_{j\in[k]}, \mathsf{R.Rec}_{\mathsf{sid}'}, \mathsf{msg}, \mathsf{Broadcast}_{\mathsf{sid}'})$.
- Inputs $(\textsc{Verify}, , \phi, \pi)$ to $\mathcal{F}_{\mathsf{NIZK}}$ and waits for a response $(\textsc{Verification}, , b)$. If $b = 0$ the message is ignored.
- After checks have been made for all the provided messages:
  - Add (S, x) to $\mathsf{Broadcast}_{\mathsf{sid}'}$.
  - For $i \in [k]$ add $(\mathsf{S}, \overline{x}_i)$ to $\mathsf{R}_i.\mathsf{Rec}_{\mathsf{sid}'}$.

If any role has more than one message with a valid signature, both should be ignored.

Execute Role: A machine M nominated for a role R should activate it for each round of the protocol until it speaks.

- If the role inputs $(\textsc{Read}, \mathsf{R}, \mathsf{S}, r')$ to $\mathcal{F}^{\mathsf{sid}}_{\mathsf{VeSPa}}$ the machine should retrieve the tuple of the form $(\mathsf{S}, \overline{x}_i)$ in $\mathsf{R.Rec}_{\mathsf{sid}}$, if no such tuple exists $\perp$ should be output directly to the role. The ciphertext should then be decrypted to obtain $x_i \leftarrow \mathsf{PKE.Dec}(\mathsf{R}.dk, \overline{x}_i)$ which may be returned to R.
- If the role inputs $(\textsc{Read}, \mathsf{S}, r')$ to $\mathcal{F}^{\mathsf{sid}}_{\mathsf{VeSPa}}$ the machine should retrieve the tuple of the form (R, x) in $\mathsf{Broadcast}_{\mathsf{sid}}$, and return x to R,

Send $\mathcal{F}_{\mathsf{VeSPa}}$: When the role R assigned to M outputs $(\textsc{Send}, \mathsf{R}, ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k)), x, w')$ to $\mathcal{F}_{\mathsf{VeSPa}}$ with session identifier sid' do the following:

- For $j \in [k]$: $\overline{x}_j \leftarrow \mathsf{PKE.Enc}(\mathsf{R}_j.ek, x_j; \rho_j)$.
- Defines the statement $\phi \leftarrow (\mathsf{R}, \mathsf{sid}', \mathsf{R}.ek, \mathcal{R}_{\mathsf{sid}'}(\mathsf{R}), (\mathsf{R}_j.ek)_{j\in[k]}, \mathsf{R.Rec}_{\mathsf{sid}'}, \mathsf{msg}, \mathsf{Broadcast}_{\mathsf{sid}'})$ and witness $w \leftarrow (\mathsf{R}.dk, (x_j, \rho_j)_{j\in[k]}, w')$
- Inputs $(\textsc{Prove}, , \phi, w)$ to $\mathcal{F}_{\mathsf{NIZK}}$ and waits for a response $(\textsc{Proof}, , \pi)$.
- Let $\mathsf{msg} = (\mathsf{R}, \mathsf{sid}', (\mathsf{R}_1, \overline{x}_1), \ldots, (\mathsf{R}_k, \overline{x}_k), x, \pi)$.
- $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{R}.sk, (\mathsf{R}, \mathsf{msg}, \pi))$.
- Input $(\textsc{Delete})$ to $\mathcal{F}_{\mathsf{RA}}$.
- Erase all private local state associated with the role R, excluding (msg, σ). In particular this includes $\mathsf{R}.dk, \mathsf{R}.sk$ and the entire state of the copy of R the machine has been running in its head.

[7] The predicate $\mathsf{KeyMatch}$ is true iff there exists randomness ρ such that $(dk, ek) \leftarrow \mathsf{KGen}(\rho)$.

– Post (msg, σ) to the bulletin board.
– Input (READY) to $\mathcal{F}_{\mathsf{RA}}$.

6.3 Security of the extended compiler

We prove the security of our extended compiler, stated in the formal theorem below.

Theorem 3. *Consider an abstract protocol Π which YOSO securely implements the ideal functionality $\mathcal{F}$ in the presence of $c/2$ static corruptions (where $c = \Omega(\kappa)$ denotes the committee size) in the $(\mathcal{F}_{\mathsf{VeSPa}}, \mathcal{F}_{\mathsf{BC\&SPP}})$-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 $\mathcal{F}_{\mathsf{RA}}$ be shorthand for $\mathcal{F}_{\mathsf{RA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{U}, 2)$ where $\mathcal{U}$ samples the uniform distribution and $\mathcal{T}(f) = c\left(1 - (1-\epsilon)(1-f)^2\right)$, for $\epsilon > 0$. Suppose Π is secure against an arbitrary number of static corruptions in the input and output roles and has the following properties* [8]*:*

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

Then, the protocol $\mathsf{Compile}(\Pi)$ UC implements the ideal functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{RA}})$-hybrid model, under the presence of $T < 0.29N$ 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 $\delta \geq 1$ and $0 < 1 - 2wf_t + wf_t^2$.

Proof. The security proof of our compiler only requires relatively light modification for our extension. Hybrids H_0 to H_2 require no modification, as they deal with modification of the role assignment and unforgeability of honest signatures.

Hybrid H_3: We again wish to transition the honest roles which the simulator is running to be honest roles in the real abstract experiment. The bulletin board ensures the consistency of messages for honest machines. The sets of messages with verifying proofs and signatures will be the same

[8] Note that all existing abstract YOSO protocols (such as the protocols in [GHK+21], [KRY22]) satisfy these properties.

across all machines, as signature verification is deterministic and $\mathcal{F}_{\mathsf{NIZK}}$ guarantees consistent verification.

We again let our simulator act as the environment toward an abstract adversary $\mathcal{A}_{\mathtt{abs}}$ of our design. The simulator specifies the static corruptions in each committee to be $\mathcal{I}_C$ as sampled previously. Our abstract adversary $\mathcal{A}_{\mathtt{abs}}$ simply takes the messages provided to it by its environment and sends them on behalf of the corrupted roles. Whenever a message is encrypted to an honest role and posted on $\mathcal{F}_{\mathsf{BB}}$ the simulator forwards the witness provided to $\mathcal{F}_{\mathsf{NIZK}}$ to recover the plaintext. The simulator then passes on the plaintext messages and the witness for $\mathcal{R}_{\mathsf{sid}}(\mathsf{R})$ to $\mathcal{A}_{\mathtt{abs}}$. When $\mathcal{A}_{\mathtt{abs}}$ receives information from $\mathcal{F}^{\mathsf{sid}}_{\mathsf{VeSPa}}$ it relays this to the simulator.

When the natural adversary reads from $\mathcal{F}_{\mathsf{BB}}$, the simulator in turn requests that $\mathcal{A}_{\mathtt{abs}}$ read all broadcast messages from the last round. The simulator may then encrypt the real messages for corrupt receivers, and use the leaked message lengths to encrypt $\mathbf{0}^{|x_{\mathsf{R}}|}$ for messages between honest roles. Note, the simulator need not provide a valid witness for these messages as it is playing the role of $\mathcal{F}_{\mathsf{NIZK}}$.

Replacing ciphertexts between honest parties is indistinguishable due to the CPA security of the encryption scheme, we do not need a decryption oracle as the adversary provides a witness containing the plaintexts to $\mathcal{F}_{\mathsf{NIZK}}$ when producing the required proof.

Hybrid H_4 may again be the same, requiring the underlying protocol to YOSO securely realise $\mathcal{F}$, thus concluding the proof.

7 Realising role assignment

In compilation, we crucially relied on the ability to program the nominations of our role assignment functionality on the fly to mitigate the adaptive corruption powers of the adversary. We will now show how to realise $\mathcal{F}_{\mathsf{RA}}$ by modifying the committee selection protocol of Benhamouda *et al.* [BGG+20] to allow equivocation of the mapping betweeen roles and machines.

We begin by recalling the high level approach of their construction. The task of choosing committee members is delegated to a nomination committee; nominators in this committee do not need to receive any private input and may therefore be self-selecting through cryptographic sortition. For a sufficiently large nomination committee the fraction of corrupt nominators will be close to the fraction of corruptions in the entire system. When a machine is chosen as a nominator it samples fresh

ephemeral keys for the role it is nominating, the public key may be broadcast along with an encryption of the secret key under a special form of anonymous PKE. As we consider an adaptive adversary with the capacity to corrupt all members of the nomination committee, were they identified, each nominator must make sure to delete its secret state prior to sending their message. All machines may then observe the broadcast channel, and attempt to decrypt each nomination ciphertext, if the decryption succeeds the machine has been nominated and can decrypt ciphertexts messages sent to the role.

To satisfy our role-assignment functionality we must make some modifications. Recall, in our simulation we want to choose the static corruptions in each committee ahead of time, only ever revealing those chosen corrupt roles. If the role assignment mechanism commits to a mapping between roles and machines a simulator may be forced to corrupt machines which have been assigned honest roles, for which it cannot equivocate. However, if the role assignment mechanism does not commit to the mapping between roles and machines this could conceivably be chosen on the fly to avoid revealing any statically honest roles. To make the approach compatible with the approach of Benhamouda *et al.* [BGG+20] we replace the encryption scheme used for nomination ciphertexts with key and message non-committing encryption (KM-NCE) [HLH+22]. We additionally introduce the use of a randomness beacon, which provides fresh uniform randomness each round, which we use to ensure the mapping from roles to nominations is uniformly random and not biased by the adversary.

Note, while KM-NCE allows equivocating for both key and message, we will only ever change the key under which ciphertexts decrypt.

Protocol Π_{RA}

Each machine M has access to a PKI containing KM-NCE public keys for each computation machine and VRF public keys. Each role stores its current long-term KM-NCE secret key as $M.sk$ and its VRF secret key as $M.sk^{\mathsf{VRF}}$. Let c be the predefined size of a committee.

New Committee: After receiving input $(\textsc{New}, \mathsf{cid}, C)$ 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 $(\textsc{Read}, r)$ to the randomness beacon, to receive randomness ρ.
- Evaluate $\mathsf{draw}, \pi \leftarrow \mathsf{Sortition}(M.sk^{\mathsf{VRF}}, (r, \mathsf{pid}, \mathsf{cid}, \rho))$.
- if draw is a winning draw proceed to nominate a party, otherwise skip the remaining steps.

- Sample a uniformly random machine index $\mathsf{pid} \leftarrow\$ \mathcal{P}$.
- 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 $c \leftarrow \mathsf{KM\text{-}NCE.Enc}(M_j.pk, (\mathsf{pid}, dk, sk))$.
- Erase the keys dk, sk and all randomness used for sampling the keys and pid, as well as any encryption randomness.
- Post $(\mathsf{cid}, ek, vk, \pi)$ 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, $(\mathsf{cid}, ek_1, vk_1, c_1, \pi_1), \ldots, (\mathsf{cid}, ek_k, vk_k, c_k, \pi_k)$.
3. Remove any elements $(\mathsf{cid}, ek_j, vk_j, c_j, \mathsf{draw}_j, \pi_j)$ posted by machine M from the list where draw_j is not a winning draw, or $\mathsf{VerifySort}(M.pk_{\mathsf{VRF}}, (r, \mathsf{pid}, \mathsf{cid}, \rho)) = \bot$
4. Sort the list lexicographically by encryption key, keeping only the c first elements. If the list does not have exactly c elements pad it with values $(\mathsf{cid}, \bot, \bot, \bot)$.
5. Input $(\textsc{Read}, r + 1)$ to the randomness beacon, to receive randomness ρ.
6. Let σ a uniformly random permutation on $[c]$ defined by the randomness ρ and apply σ to the list.
7. Loop over the list, for the jth element $(\mathsf{cid}, ek_j, vk_j, c_j)$:
   - Append (ek_j, vk_j, C_j) to PKeys.
   - Attempt to decrypt $(\mathsf{pid}, dk, sk) \leftarrow \mathsf{KM\text{-}NCE.Dec}(M_j.sk, c_j)$. If $(\mathsf{pid}, dk, sk) \neq \bot$ and pid matches the machine which posted the element to the bulletin board, append $(\mathsf{pid}, dk, sk, C_j)$.
8. 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 $M.sk$.

Ready: When given input $\textsc{Ready}$, generate a new key pair $(pk, sk, tk) \leftarrow \mathsf{KM\text{-}NCE.Gen}()$, setting $M.sk = sk$ and deleting tk immediately. Finally, post (pid, pk) to the bulletin board.

We now prove the security of our role assignment mechanism. The protocol ensures at most $\mathcal{T}(f) = c\left(1 - (1 - \epsilon)(1 - f)^2\right)$ of the c roles in a committee are assigned to corrupt machines when the committee is finished being nominated. Here f is the fraction of corruptions at the point where the committee finishes being nominated. Intuitively this corresponds to guaranteeing that the remaining $(1 - f)N$ honest machines have nominated other machines which have remained honest at least a fraction $(1 - f)$ of the time.

Theorem 4. *For threshold function* $\mathcal{T}(f) = c\left(1-(1-\epsilon)(1-f)^2\right)$ *and the uniform distribution* $\mathcal{U}$*. If the* KM-NCE *scheme used has* $\mathsf{KMNC_k}$-CCA *(for* $\mathsf{k} = \mathsf{poly}(\kappa)$ [9] *) and* KM-NCE-UR *security and the sortition has winning probability* $c/((1+\epsilon')N)$ *for* $\epsilon' > 0$*. Then, the protocol* Π_{RA} *UC realises the functionality* $\mathcal{F}_{\mathsf{RA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{U}, 2)$ *in the presence of* $t \leq N$ *adaptive corruptions in the* $(\mathcal{F}_{\mathsf{Beacon}}, \mathcal{F}_{\mathsf{BB}})$*-hybrid model*

Proof. We prove the role assignment protocol described in Section 7 securely implements the $\mathcal{F}_{\mathsf{RA}}(\mathcal{P}, c, \mathcal{T}, \mathcal{U}, 2)$ functionality, by a sequence of indistinguishable hybrids, arriving at a simulator $\mathcal{S}_{\mathsf{RA}}$.

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

We present a simulator for a real world adversary $\mathcal{A}_{\mathsf{RA}}$. The simulator interacts with the ideal functionality:

- When an honest machine given $(\textsc{New}, \mathsf{cid}, C)$ wins the sortition,
  - With probability $\frac{|\mathcal{H}|}{|\mathcal{P}|}$ input $(\textsc{Nom-Honest}, \mathsf{cid})$ to $\mathcal{F}_{\mathsf{RA}}$. Upon receiving $(\textsc{Nom-Honest}, \mathsf{cid}, ek, vk)$ post $(\mathsf{cid}, ek, vk, c)$ to the bulletin board, where c is a faked ciphertext.
  - Otherwise, choose a uniformly random corrupt party $\mathsf{pid} \leftarrow \mathcal{I}$, and sample fresh keys giving input $(\textsc{Nom-Corrupt}, \mathsf{cid}, \mathsf{pid}, (ek, vk), (dk, sk))$ to $\mathcal{F}_{\mathsf{RA}}$. Post $(\mathsf{cid}, ek, vk, c)$ to the bulletin board, where c is a real encrption of the keys.
- When $\mathcal{A}_{\mathsf{RA}}$ posts ciphertext $(\mathsf{cid}, ek, vk, c)$ to the bulletin board, the simulator attempts to decrypt c for each honest machine. If $(\mathsf{pid}^*, dk, sk) = \mathsf{KM\text{-}NCE.Dec}(M_j.sk, c_j)$, where pid^* is the identifier for the corrupt machine which posted the message, then input $(\textsc{Nominate-Corrupt}, \mathsf{pid}, (ek, vk), (dk, sk))$ where pid is the identifier for M_j. Abort, if the ciphertext correctly decrypts for more than one honest party. If it only decrypts to $\perp$ for honest parties, input $(\textsc{Nom-Corrupt}, \mathsf{cid}, \perp, (ek, vk), (\perp, \perp))$ to $\mathcal{F}_{\mathsf{RA}}$.
- When $\mathcal{A}_{\mathsf{RA}}$ 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 $(\textsc{Corrupt-Nominee}, \mathsf{cid}, \mathsf{pid})$ to the functionality to receive secret keys for the role in $(\textsc{Corrupt-Nominee}, \mathsf{cid}, \mathsf{pid}, dk, sk)$. 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.
- When proceeding to the next round the simulator inputs $(\textsc{Nom-Corrupt}, \mathsf{cid}, \perp, (\perp, \perp), (\perp, \perp))$ to $\mathcal{F}_{\mathsf{RA}}$ until each committee being nominated in the current round has exactly C roles. This causes the

[9] To weaken this to $\mathsf{k} = O(1)$ would require a bound on the number of honest nominations a machine could receive before refreshing its key

functionality to finish, outputting (FINISH, $\mathsf{cid}, \phi, \mathsf{PKeys}$), where ϕ fixes the permutation between keys and roles. In the subsequent round the simulator should then reprogram the randomness beacon to match ϕ.

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.

Hybrid H_0*:* Run the real experiment $REAL_{\Pi_{\text{nat}},\mathcal{A}_{\text{nat}},\mathcal{E}_{\text{nat}}}(1^\kappa)$.

Hybrid H_1*:* The same as previous, except the simulator aborts if the corrupt parties wins the sortition more than $(1+\epsilon)fc$ times for a given committee.

Hybrid H_2*:* Abort if any honest machine receives more than k nominations from honest parties between receiving READY and the subsequent DELETE.

Hybrid H_3*:* Abort the experiment if the adversary ever broadcasts $(\mathsf{cid}, ek, vk, c, \pi)$ where more than one honest machine can decrypt c to something different to $\perp$.

Hybrid H_4*:* For each encryption 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.

Hybrid H_5*:* 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 $\frac{1}{|\mathcal{H}|}$. When a role is assigned to the machine input corrupt nominee 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_6*:* When an honest machine is given $(\textsc{New}, \mathsf{cid}, C)$, with probability s if the machine has wins the sortition do the following:

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

Hybrid H_7*:* 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 c. The simulator aborts if too many nomination queries are made. We are now in the ideal world.

We now show the indistinguishability of the hybrids above.

$\mathsf{H}_0 \approx \mathsf{H}_1$*:* An adversary winning the sortition too often, given the random seed from $\mathcal{F}_{\mathsf{Beacon}}$, may be reduced to an adversary breaking the pseudo-randomness property of the sortition, by predicting which honest parties will win.

$\mathsf{H}_1 \approx \mathsf{H}_2$*:* The hybrid may only abort if there are at least k collisions between the honestly nominated role. 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.

$\mathsf{H}_2 \approx \mathsf{H}_3$*:* The indistinguishability of these hybrids follows directly from the unique recipient property of the KM-NCE scheme.

$\mathsf{H}_3 \approx \mathsf{H}_4$*:* 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 tollerated by the scheme. An adversary distinguishing the two hybrids would win the KMNC-CCA game with the same advantage.

$\mathsf{H}_4 \approx \mathsf{H}_5$*:* Sampling the assignment of roles to machines lazily induces the same probability distribution as if done beforehand.

$\mathsf{H}_5 \approx \mathsf{H}_6$*:* The honest machines sample the keys identically to $\mathcal{F}_{\mathsf{RA}}$, and nominate an honest party with probability $|\mathcal{H}|/N$

$\mathsf{H}_6 \approx \mathsf{H}_7$*:* We wish to upper bound the number of corrupt nominations in a committee, to do this we will lower bound the number of remaining honest parties. First, we upper bound the number of machines winning the sortition by considering the random variable $X \sim \mathsf{Bin}(N, s)$, where s is the winning probability for a given party. By a Chernoff bound we know

$$\Pr[X > (1+\epsilon_1)Ns] < \exp\left(-\frac{Ns\epsilon_1^2}{(2+\epsilon_1)}\right).$$

For $Ns = \Omega(\kappa)$ and a fixed ϵ_1 this probability will be negligible in the security parameter. We let our committee size $c = (1+\epsilon_1)Ns$, and will now bound the number of corrupt nominations. We have ensured that there are never more than c total nominations, ensuring that no nominations are ever ignored. Thus, we only need to lower bound the fraction of honest parties in our 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.

Let $H \sim \mathsf{Bin}(|\mathcal{P}|(1-f), (1-f)s)$ be the random variable describing the number of honest nominations. If we can lower bound H we may upper bound the number of corrupt roles in a committee. We wish to ensure,

$$\Pr\left[c - H > c\left(1 - (1-\epsilon)(1-f)^2\right)\right] = negl(\kappa).$$

Which, for $\epsilon_1, \epsilon_2 > 0$ may be equivalently expressed as

$$\begin{aligned}\Pr\left[c\left(\frac{1-\epsilon_2}{1+\epsilon_1}(1-f)^2\right) > H\right] &= \Pr\left[(1-\epsilon_2)Ns(1-f)^2 > H\right] \\ &< \exp\left(-\frac{N(1-f)^2 s\epsilon_2}{2}\right)\end{aligned}$$

Which, as previously, is negligible for $Ns = \Omega(\kappa)$ and a fixed ϵ_2. It is therefore indistinguishable if we enforce the threshold function $\mathcal{T}(f)$.

8 The versatility of our compiler

The compiler we present allows the compilation of YOSO protocols using both $\mathcal{F}_{\mathsf{BC\&SPP}}$ and $\mathcal{F}_{\mathsf{VeSPa}}$. Of the existing literature only Kolby *et al.* present computationally secure protocols in the $\mathcal{F}_{\mathsf{VeSPa}}$-hybrid model [KRY22], having introduced the functionality. However, existing works which make non-black-box use of the communication between roles may be recast into the $\mathcal{F}_{\mathsf{VeSPa}}$-hybrid model allowing for their efficient compilation.

We provide one such example. Braun *et al.* construct a YOSO MPC protocol from class groups, following the circuit based CDN paradigm of [CDN01]. Their protocol proceeds by first performing a distributed key generation to obtain a key for a threshold linearly homomorphic encryption scheme, which is then used for the circuit evaluation.

In the construction of their protocol they assume access to explicit public keys allowing them to prove statements about the ciphertexts and public messages with NIZK. The NIZK proofs are used in three of their functionalities, $\mathsf{CreateVSS}$, $\mathsf{CreateTriple}$ and $\mathsf{YOSO-ABB}$. Proving the exact same relations about the messages sent through $\mathcal{F}_{\mathsf{VeSPa}}$ would clearly preserve security, giving the simulator access to the same witnesses it could extract from explicit proofs.

Braun *et al.* [BDO22] specifically tailor their statements to have efficient proofs for the class group encryption scheme they use [CCL+19]. As our extended compiler is secure for any PKE scheme with CPA security, it could in particular be instantiated with the same class group scheme preserving their efficiency.

References

AHKP22. Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 502–531. Springer, Heidelberg, November 2022.

BDO22. Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. Cryptology ePrint Archive, Report 2022/1437, 2022. `https://eprint.iacr.org/2022/1437`.

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, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, November 2020.

CCL+19. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. Cryptology ePrint Archive, Report 2019/503, 2019. https://eprint.iacr.org/2019/503.

CDN01. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

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

EHK+13. Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013.

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. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.

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 2021, Part III*, volume 13044 of *LNCS*, pages 32–61. Springer, Heidelberg, November 2021.

GOS12. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012.

HLH+22. Zhengan Huang, Junzuo Lai, Shuai Han, Lin Lyu, and Jian Weng. Anonymous public key encryption under corruptions. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 423–453. Springer, Heidelberg, December 2022.

KMTZ13. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

KRY22. Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Constant-round yoso mpc without setup. Cryptology ePrint Archive, Paper 2022/187, 2022. https://eprint.iacr.org/2022/187.

MRV99. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.

Nie02. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002.