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, Boston, USA
canetti@bu.edu
[2] Aarhus University, Aarhus, Denmark
{sk,divya,sophia.yakoubov}@cs.au.dk
[3] Technology Innovation Institute, Abu Dhabi, 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.

© International Association for Cryptologic Research 2023
G. Rothblum and H. Wee (Eds.): TCC 2023, LNCS 14370, pp. 33–62, 2023.
https://doi.org/10.1007/978-3-031-48618-0_2

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 Sect. 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 a random 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 wishes to delete its state before it speaks on behalf of a role, and (c) when a machine is unavailable for nomination while it refreshes its secret state.[1] The formal details appear in Sect. 3.

1.1.2 Compiling Abstract Protocols In Sect. 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

[1] In our particular use-case, machines are unable to be nominated between deleting their previous secret key and broadcasting a fresh public key. This allows one machine to hold multiple roles, but prevents nominations which overlap with the machine speaking for a role.

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. Just before a machine speaks on behalf of a role, it instructs the $\mathcal{F}_{\mathsf{RA}}$ to delete its state. After speaking, it instructs $\mathcal{F}_{\mathsf{RA}}$ that it is ready for new nominations.

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 $\mathcal{F}_{\mathsf{RA}}$ may be suitably re-programmed so that adaptive corruption of machines are appropriately matched to the already chosen static corrupt roles. We refer to Sect. 5 for details on the technicalities in our proof.

Compiling Abstract Protocols that Require Message Verification. The above compiler supports abstract protocols that use only ideally private point-to-point and broadcast channels. This does not cover a large class of abstract YOSO protocols where parties are expected to accompany their messages with zero-knowledge proofs that relate their outgoing messages to their secret state and previously received messages. Indeed, in order to compile such protocols to natural ones, such proofs would need to involve both secret state from the abstract protocol and secret keys from the compiler itself. In Sect. 6, we show how our compiler can be extended to abstract protocols that contain such constructs. 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 Sect. 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 became 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 ($\mathsf{KM\text{-}NCE}$) from [HLH+22], which is an extension of receiver non-commiting encryption. Informally, a $\mathsf{KM\text{-}NCE}$ 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 for any public key, by generating an appropriate secret key on the fly. We briefly recall the syntax of a KM-NCE scheme, referring the reader to [HLH+22] for a more detailed motivation.

$\mathsf{Setup}(1^\kappa) \rightarrow \mathsf{pp}$**:** Given security parameter 1^κ, the setup algorithm generates public parameters pp.

$\mathsf{Gen}(\mathsf{pp}) \rightarrow (pk, sk, tk)$**:** Given public parameters pp, the key generation algorithm produces a public key pk and secret key sk, as well as a trapdoor key tk. The trapdoor key is not used for encryption or decryption, but instead provides additional information for the purposes of opening simulated ciphertexts.

$\mathsf{Enc}(\mathsf{pp}, pk, m) \rightarrow c$**:** Given public parameters pp, public key pk and a message m, the encryption algorithm produces a ciphertext c.

$\mathsf{Dec}(\mathsf{pp}, sk, c) \rightarrow m$**:** Given public parameters pp, public key pk and a ciphertext c, the decryption algorithm outputs a plaintext m.

$\mathsf{Fake}(\mathsf{pp}) \rightarrow (c, \tau)$**:** Given only the public parameters pp, the fake algorithm produces a fake ciphertext c and additional trapdoor information τ.

$\mathsf{Open}_k(\mathsf{pp}, tk, pk, sk, (c^*_\gamma, \tau^*_\gamma, m^*_\gamma)_{\gamma \in [k]}) \rightarrow sk'$:] Given public parameters pp, keys tk, pk, sk, and k tuples, each containing a ciphertext c_γ, its trapdoor information τ_γ and a desired plaintext m_γ the open algorithm produces a fresh secret key sk' corresponding to pk, such that each ciphertext appropriately decrypts to the desired plaintext.

In the security experiments for KM-NCE the adversary is never given trapdoor keys, implicitly requiring secure erasure of these keys if we wish to achieve adaptive security.

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* $\mathbf{Adv}^{\mathsf{KM\text{-}NCE\text{-}CCA}}_{\mathsf{KM\text{-}NCE},\mathcal{A},k}(\lambda) :=$

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

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

```
$\mathbf{Exp}^{\text{KM-NCE-CCA-real}}_{\text{KM-NCE},\mathcal{A},k}(\lambda)$:
$\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$
$(pk, sk, tk) \leftarrow\$ \mathsf{Gen}(\mathsf{pp})$
$((m^*_\gamma)_{\gamma\in[k]}, \mathsf{state}_1) \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{Dec}}}(\mathsf{pp}, pk)$
$(c^*_\gamma \leftarrow\$ \mathsf{Enc}(\mathsf{pp}, pk, m^*_\gamma))_{\gamma\in[k]}$
$\mathsf{state}_2 \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{Dec}}}((c^*_\gamma)_{\gamma\in[k]}, \mathsf{state}_1)$
$b \leftarrow\$ \mathcal{A}_3(sk, \mathsf{state}_2)$
Return $b$

$\mathcal{O}_{\mathsf{Dec}}(c)$:
If $c \in \{c^*_\gamma : \gamma \in [k]\}$: Return $\perp$
$m = \mathsf{Dec}(\mathsf{pp}, sk, c)$
Return $m$

$\mathbf{Exp}^{\text{KM-NCE-CCA-ideal}}_{\text{KM-NCE},\mathcal{A},k}(\lambda)$
$\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$
$(pk, sk, tk) \leftarrow\$ \mathsf{Gen}(\mathsf{pp})$
$((m^*_\gamma)_{\gamma\in[k]}, \mathsf{state}_1) \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{Dec}}}(\mathsf{pp}, pk)$
$((c^*_\gamma, \tau^*_\gamma) \leftarrow\$ \mathsf{Fake}(\mathsf{pp}))_{\gamma\in[k]}$
$\mathsf{state}_2 \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{Dec}}}((c^*_\gamma)_{\gamma\in[k]}, \mathsf{state}_1)$
$sk' \leftarrow\$ \mathsf{Open}_k(\mathsf{pp}, tk, pk, sk, (c^*_\gamma, \tau^*_\gamma, m^*_\gamma)_{\gamma\in[k]})$
$b \leftarrow\$ \mathcal{A}_3(sk', \mathsf{state}_2)$
Return $b$
```

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

Note, $\mathsf{KMNC_k}$-CCA security implies conventional adaptive CCA security, as the fake algorithm does not take a message as input. By a hybrid argument, the encryption of any message m_0 must be indistinguishable from a faked ciphertext, which in turn is itself indistinguishable from the encryption of any other message m_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.

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* $\mathcal{A}$, $\Pr[\mathbf{Exp}^{\text{KM-NCE-UR}}_{\text{KM-NCE},\mathcal{A}}(\lambda) = 1]$ *is negligible, where* $\mathbf{Exp}^{\text{KM-NCE-UR}}_{\text{KM-NCE},\mathcal{A}}$ *is defined in Fig. 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

```
$\mathbf{Exp}^{\mathsf{KM\text{-}NCE\text{-}UR}}_{\mathsf{KM\text{-}NCE},\mathcal{A}}(\lambda)$:
$\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$
$((pk_i, sk_i, tk_i) \leftarrow\$ \mathsf{Gen}(\mathsf{pp}))_{i\in[h]}$
$c \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{Dec}},\mathtt{RO}}(\mathsf{pp}, \{pk_i\}_{i\in[h]})$
If $\exists i_1, i_2 \in [h] : i_1 \neq i_2 \wedge$
$\mathsf{Dec}(\mathsf{pp}, sk_{i_1}, c) \neq \perp \wedge$
$\mathsf{Dec}(\mathsf{pp}, sk_{i_2}, c) \neq \perp$, return 1.
Otherwise, return 0.

$\mathcal{O}_{\mathsf{Dec}}(c)$:
If $c \in \{c^*_\gamma : \gamma \in [k]\}$: Return $\perp$
$m/\perp \leftarrow \mathsf{Dec}(\mathsf{pp}, sk, c)$
Return $m/\perp$

$\mathtt{RO}(s)$:
$\mathcal{S}$ returns a uniformly random $t$.
```

Fig. 2. The unique recipient experiment.

(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, $\widetilde{\mathsf{Pub}}$) 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 public parameters $\mathsf{pp} = (\mathbb{G}, g, q)$.

- $(pk, sk, \emptyset) \leftarrow\$ \ \mathsf{Gen}(\mathsf{pp})$: 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. We observe that an adversary querying these values it may be used to solve the computational 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 greater than or 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 through the use of a Verifiable Random Function (VRF) [MRV99]. Importantly, a party can find out whether it was selected through local computation, given the output from the VRF.

Usual VRF definitions guarantee output unpredictability for adversarially chosen inputs, provided that the keys were honestly generated. In our setting this is insufficient, as it does not preclude an adversary choosing malformed keys which bias its output distribution, causing it to be selected more frequently. To ensure security against rogue key attacks of this form we will use the functionality $\mathcal{F}_{\mathsf{VRF}}$ from [DGKR18], which explicitly allows malicious key generation and VRF evaluation. The key property on which we will rely is "unpredictability under malicious key generation". This property is captured by the functionality always sampling the VRF output regardless of whether the specified key was maliciously generated. For a complete description of $\mathcal{F}_{\mathsf{VRF}}$ with a corresponding realisation we refer the reader to [DGKR18].

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 builds on top of the plain UC model. In particular, it uses the following constructs:

– Parties in the UC framework represent *roles,* namely abstract responsibilities. In an actual execution of a YOSO protocol, the roles will would be carried out by machine to which they are assigned to on the fly. The design of a YOSO protocol 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.

As in [GHK+21], (and following [KMTZ13]) we use a bounded-delay broadcast functionality, along with a global clock, to capture synchronous communication. We recall the ideal functionality allowing point-to-point and broadcast communication as in [GHK+21].

Functionality $\mathcal{F}_{\mathsf{BC\&SPP}}$ [GHK+21]

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 $(\textsc{Send}, \mathsf{S}, ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k)), x)$ 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 the simulator $\mathcal{S}$.
  - For corrupt roles R_i output x_i to the simulator $\mathcal{S}$. *Leak messages lengths and the broadcast message to the simulator in a rushing fashion.*
  - If S is honest give $\textsc{Spoke}$ to S.
- On input $(\textsc{Read}, \mathsf{R}, \mathsf{S}, r')$ in round r where $r' < r$ for $x = y(r', \mathsf{S}, \mathsf{R})$ output x to R.
- On input $(\textsc{Read}, \mathsf{S}, r')$ 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 $\mathcal{F}_{\mathsf{RA}}$[2], 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 our $\mathcal{F}_{\mathsf{RA}}$ 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 determined 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. This command allows accounting for the corruptions performed during the nomination process as needed, rather than always having to generate corrupt keys in proportional to the worst case threshold.

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 M to:

1. *'Read'*: this allows it to retrieve public keys corresponding to the roles that have been assigned, as well as to obtain secret keys if it has been assigned a role.
2. *'Delete'*: this command revokes M's ability to perform future reads until the point where it inputs *'Ready'*. (This revocation will also enable the implementation protocol to erase any secret keys that allow M to read information related to already assigned roles.)
3. *'Ready'*: this allows it to signal that it is available to be assigned a new role. We maintain both a global set of ready machines ("ready set"), and a committee-specific ready set. The latter keeps track of machines that have been ready throughout the nomination process for that committee.

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

If a machine that has been assigned a role gets corrupted after it has retrieved its secret keys (which it learns when it inputs 'read') but before it inputs 'delete', its secret keys are leaked to the adversary. However, if it gets corrupted after it inputs 'delete', its secret keys remain hidden. As we will see later, this is crucial for adaptive security, as it allows us to argue that an adversary gets no advantage in corrupting a role after its execution.

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 present $\mathcal{F}_{\mathsf{RA}}$ as a functionality which is reused for multiple committees rather than the perhaps simpler approach of a one time functionality for each committee. We justify this choice by considering how existing constructions update their PKI. Specifically, whenever a machine has held a role and subsequently revealed itself, said machine must refresh its long term keys. This renders the machine unable to decrypt earlier messages pertaining to the revealed role. These key erasures and updates to the PKI impede treating it as a global setup (see [CDPW07]), which would allow consolidating these to just a single PKI. Using a single $\mathcal{F}_{\mathsf{RA}}$ for multiple committees thus forces any realisation to deal with this challenge of updates directly.

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, namely it has access to global clock functionality as in the model of Katz *et al.* [KMTZ13]. 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 current fraction of corrupt machines).
- $\mathcal{D}$ denoting a sampling algorithm, and
- $\mathtt{delay}$ denoting the upper bound on the number of rounds required to complete nomination.

Init: The functionality is notified by the adversary whenever a party is corrupted/ restored, and maintains the current partition of $\mathcal{P}$ into the 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 up until the round r specified by the cid [a], store $(\mathsf{cid}, C, \mathsf{PKeys} = \emptyset, \mathsf{SKeys} = \emptyset, \mathtt{cor} = 0, \mathtt{nom} = 0, fin = \perp)$. Ignore the command if any value is already stored for cid.

- The lists PKeys and SKeys are initially empty. The list PKeys would be updated with tuples (ek, vk, R) where (ek, vk) refer to the public keys established for a role R. The list SKeys would be updated with tuples $(\mathsf{pid}, dk, sk, \mathsf{R})$ where (dk, sk) refer to the secret keys corresponding to the role R, which has been assigned to machine with identifier pid.
- The corruption and nomination counters, $\mathtt{cor}$ and $\mathtt{nom}$, 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 the simulator $\mathcal{S}$.

[a] For simplicity of exposition, we consider the case where all honest parties are expected to take part in each assignment of a role. A natural relaxation would only require some minimal quorum of parties to participate.

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 the simulator $\mathcal{S}$, retrieve the value $(\mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys}, \mathtt{cor}, \mathtt{nom}, fin)$. If no such value exists do nothing. If $\mathtt{nom} < c$, do the following:

- Update $\mathtt{nom} \leftarrow \mathtt{nom} + 1$.
- 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 $\mathtt{nom} = c$, go to procedure $\mathsf{Finish}(\mathsf{cid})$.
- Output $(\textsc{Nom-Honest}, \mathsf{cid}, ek, vk)$ to the simulator $\mathcal{S}$.

Nominate corrupt: On input $(\textsc{Nom-Corrupt}, \mathsf{cid}, \mathsf{pid}, (ek, vk), (dk, sk))$ from the simulator $\mathcal{S}$, retrieve the value $(\mathsf{cid}, C, \mathsf{PKeys}, \mathsf{SKeys}, \mathtt{cor}, \mathtt{nom}, fin)$. 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 $(\textsc{Corrupt-Nominee}, \mathsf{cid}, \mathsf{pid})$ from the simulator $\mathcal{S}$, retrieve the value $(\mathsf{cid}', C, \mathsf{PKeys}, \mathsf{SKeys}, \mathtt{cor}, \mathtt{nom}, fin)$ where $\mathsf{cid} = \mathsf{cid}'$. 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 $(\mathsf{pid}', dk, sk, \perp)$ uniformly at random between the values of SKeys where $\mathsf{pid}' = \perp$.
- Update this value to be $(\mathsf{pid}, dk, sk, \perp)$
- Output $(\textsc{Corrupt-Nominee}, \mathsf{cid}, \mathsf{pid}, dk, sk)$ to the simulator $\mathcal{S}$.

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

- Sample a random permutation ϕ on $[c]$.
- For the ith element of PKeys update $(ek, vk, \perp)$ to $(ek, vk, C_{\phi(i)})$.
- For the ith element of SKeys update $(\mathsf{pid}, dk, sk, \perp)$ as follows:
  - If $\mathsf{pid} = \perp$, choose an honest machine uniformly at random as $\mathsf{pid}' \leftarrow\$ \mathcal{D}(\mathcal{H}, \mathcal{P})$. If $\mathsf{pid}' \in \mathsf{Ready}_{\mathsf{cid}}$, update to $(\mathsf{pid}', dk, sk, C_{\phi(i)})$.
  - Else, update to $(\mathsf{pid}, dk, sk, C_{\phi(i)})$.
- Let r' the current round number (read from the global clock). Set $fin = \top$ for cid if $r' \leq r + \mathtt{delay}$ (where r denotes the round number specified by the cid).

Output $(\textsc{Finish}, \mathsf{cid}, \phi, \mathsf{PKeys})$ to the simulator $\mathcal{S}$ when finished.

Read: On input $(\textsc{Read}, \mathsf{cid})$ from M with identifier pid, retrieve the value $(\mathsf{cid}^*, C, \mathsf{PKeys}, \mathsf{SKeys}, \mathtt{cor}, \mathtt{nom}, fin)$ where $\mathsf{cid} = \mathsf{cid}^*$ and $fin = \top$. If no such value exists, or M has read the output of committee cid before, do nothing.

- Collect all values $(\mathsf{pid}^*, dk, sk, \mathsf{R})$ in SKeys where $\mathsf{pid}^* = \mathsf{pid}$ into a list SKeys'.
- 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 $(\mathsf{pid}^*, \perp, \perp, \mathsf{R})$. Disallow any future signing queries by M for role 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 $fin = \perp$.
- 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. Output $(\textsc{Ready}, \mathsf{pid})$ to the simulator $\mathcal{S}$.

Corrupt: Upon receiving $(\textsc{Corrupt}, \mathsf{pid})$ 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 Sect. 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 $m(r, M) = \mathsf{msg}$. *Store the broadcast message from the role.*
  - Output $(\textsc{Send}, \mathsf{sid}, \mathsf{msg})$ to the simulator $\mathcal{S}$.
- 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 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 it 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 $\mathsf{Compile}(\Pi)$

Notation: The algorithm $\mathsf{Nominate}(r, \{\mathsf{Broadcast}_{\mathsf{sid}}\}_{\mathsf{sid} \in \mathsf{SID}})$ denotes a publicly computable function which when given a round number and public state outputs the set of committees $\{\mathsf{cid}_i, C_i\}_{i \in [k]}$ to be nominated in that particular round. We assume that all the cid_i's contains the round number r.

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 $\mathsf{R} \in \mathsf{Role}^{\mathrm{IN}} \cup \mathsf{Role}^{\mathrm{OUT}}$, 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 (i.e. as per the reading of the global clock), 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 $(\textsc{Read}, \mathsf{cid})$ 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 (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 $\mathsf{R} \in \mathsf{Role}^{\textsc{Comp}}$.

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 $\mathsf{S} \in \mathsf{Role}^{\textsc{In}} \cup \mathsf{Role}^{\textsc{Out}}$
[b] Here, signatures can be avoided if $\mathsf{R} \in \mathsf{Role}^{\textsc{In}}$.

5 Security of the Compiler

In this section, we prove the security of the compiler presented in Sect. 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 our 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).

To rely on the static simulator of our abstract protocol it is essential that the natural world adversary cannot influence which roles are revealed through its chosen corruptions of machines. As a starting point, let us consider what goes wrong if a natural simulator is forced to commit to a mapping from roles to machines. An adaptive adversary might then subsequently choose which machines to corrupt based on this commitment. The simulator is essentially forced to guess which machines the adversary will corrupt making it overwhelmingly likely to fail.

To circumvent this issue we may instead consider the possible simulation strategy if our simulator were not committed to this role to machine mapping. Our static abstract simulator must always fix a choice of corrupt roles. The state of these corrupt roles may be simulated, making it acceptable to assign them to corrupt machines. Conversely, we have no way to simulate the state of honest roles, so these must never be revealed to the adversary. During simulation, the simulator presents a role assignment functionality to the natural world adversary.

The natural world adversary expects the roles to be assigned to the machines it has corrupted in proportion to its expended corruptions. This may easily be accounted for by sampling a mapping where an appropriate number of statically corrupt roles are assigned to these machines. Things get more challenging when we start to consider adaptive corruptions, in the real world the adversary will sometimes get lucky and corrupt a machine which has been assigned a role. If we simply fix the mapping from roles to machines at the time of nomination this could cause simulation to fail if the newly corrupted machine had been assigned an honest role. However, if our role assignment functionality does not leak anything to the adversary about the mapping of honest roles we may simply change the assignment of this honest role to a machine which remains honest. This will of course affect the number of roles revealed to the adversary, to account for this we must additionally maintain some budget of statically corrupt roles, which we reveal in place of the honest roles.

As the simulator now controls which roles are revealed to the adversary it may be sure that it never has to open a ciphertext sent between the holders of two honest roles. As a result these ciphertexts need not be non-committing, allowing the use of the much more efficient CCA secure encryption.

We define the class of protocols which are compatible with our compiler.

Definition 3 (Compiler compatible protocol). *We call a protocol Π a compiler compatible secure implementation of $\mathcal{F}$ with threshold c/w, if the following conditions are satisfied:*[6]

- *Let $c = \Omega(\kappa)$ denote the committee size. Then, Π must YOSO securely implement the ideal functionality $\mathcal{F}$ in the presence of c/w static corruptions in the computation committees and an arbitrary number of static corruptions in the input and output roles.*
- *All honest roles in the same committee speak in the same round.*
- *There exists a positive constant* `delay`*, such that it is publicly computable which committees need to be nominated at least* `delay` *round(s) in advance.*
- *There exists a constant $R_{max} \geq \kappa$, denoting 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).*

Theorem 2. *Consider an abstract protocol Π in the $\mathcal{F}_{\mathsf{BC\&SPP}}$-hybrid model, which is a compiler compatible secure implementation of $\mathcal{F}$ with threshold c/w (Definition 3). 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 a function $\mathcal{T}(f)$. Further, assume the schemes* PKE *and* SIG *used by $\mathcal{F}_{\mathsf{RA}}$ are adaptive IND-CCA and EUF-CMA secure respectively.*

Then, assuming a PKI setup, the protocol $\mathsf{Compile}(\Pi)$ *UC implements the ideal functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{RA}}, \mathcal{F}_{\mathsf{VRF}})$-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*

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

constant $1 \leq \delta$ *and* f_t *is fixed such that there exists a constant* $\epsilon > 0$ *where for all* $0 < f < f_t$ *it holds that* $\mathcal{T}(f) + (1+\epsilon)(f_t - f)c < c/w$.

If we apply Theorem 2 to the threshold function achieved by our role assignment protocol in Sect. 7 we obtain the following corollary.

Corollary 1. *For* $\mathcal{T}(f) = c\left(1 - (1-\epsilon)(1-f)^2\right)$ *a protocol tolerating* c/w *corruptions may be compiled to a protocol tolerating* $T < Nf_t$ *adaptive corruptions, where* f_t *satisfies* $0 < 1 - 2wf_t + wf_t^2$.[7]

We refer the reader to the full version of this paper for a proof of Theorem 2.

6 Compiling Abstract Protocols Requiring Verification

Our compiler in Sect. 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.

Before showing how our compiler may be extended to protocols in the $(\mathcal{F}_{\mathsf{VeSPa}}, \mathcal{F}_{\mathsf{BC\&SPP}})$-hybrid model we will first reflect on the broader role of message verifiability within YOSO protocols. When using $\mathcal{F}_{\mathsf{BC\&SPP}}$ all point-to-point messaging is ideal, making it impossible to directly provide verifiability guarantees for any single message in a single round. Works studying information theoretic YOSO MPC [GHK+21, DKI+23] achieve verifiability by constructing verifiable secret sharing (VSS) protocols in the abstract world. They then make use of VSS to construct their desired MPC protocols. These protocols explicitly handle their need for verifiable message passing in the abstract world, and thus inherit these same guarantees when compiled to the natural world. There are drawbacks to this approach of explicit abstract world verifiability, as existing VSS constructions all introduce an overhead in both rounds and a number of intermediate roles.

An alternative approach follows from the ideas within computationally secure protocols, where verifiability may come from non-interactive zero-knowledge

[7] This holds when $f_t < 1 - \frac{\sqrt{w^2 - w}}{w}$. For $w = 2$, namely when the abstract protocol withstands honest minority, this allows $f_t \approx 0.29$. For $w = 1.1$, namely when the abstract protocol withstands corruption of roughly 90% of the parties, this allows $f_t \approx 0.7$.

proofs, rather than additional interaction. In the context of YOSO the restriction to $\mathcal{F}_{\mathsf{BC\&SPP}}$ means that we only consider black box communication, and thus cannot directly prove statements about point-to-point messages. To resolve the limitation Kolby *et al.* [KRY22] introduced a new *verifiable state propagation* functionality which enabled enforcing statements for point-to-point messages, giving verifiability. A natural question to consider is whether it is possible to realise $\mathcal{F}_{\mathsf{VeSPa}}$ in the abstract world given $\mathcal{F}_{\mathsf{BC\&SPP}}$. However, if we recall the cost of achieving VSS in the $\mathcal{F}_{\mathsf{BC\&SPP}}$-hybrid model, our hopes of verifying more complex relations, without a significant round and communication complexity overhead are quickly dampened. Conversely, if we do not realise $\mathcal{F}_{\mathsf{VeSPa}}$ we are left with a protocol which remains incompatible with compilation. This leaves us with a choice of either realising $\mathcal{F}_{\mathsf{VeSPa}}$ in the abstract world, or adapting our compiler to produce protocols which enforce the guarantees of $\mathcal{F}_{\mathsf{VeSPa}}$, essentially making verifiability explicit during the translation to the natural world.

We observe that our compiler is actually well suited to the addition of message verifiability, making this a desirable choice. Recall, our modifications have eliminated the need for non-committing encryption for protocol messages, instead simply requiring CCA security. If we extend the few requirements we make of our encryption scheme to additionally permitting efficient proofs of knowledge of plaintext, we may use non-interactive zero-knowledge to prove that the encrypted messages between roles satisfy whatever relations we require.

6.1 Verifiable State Propagation

In this section, we recall 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).

In more detail, a sender role S in the abstract protocol invokes $\mathcal{F}_{\mathsf{VeSPa}}$ with the following information: (a) the point-to-point messages S intends to send to a set of recipient roles (b) the messages S intends to broadcast (c) witness (comprising of the internal state of S such as its private randomness used to compute its outgoing messages).

Consider the statement comprising of these outgoing point-to-point (say, ϕ_{send}) and broadcast messages (say, $\phi_{broadcast}$), the incoming messages that were received by S (say, $\phi_{receive}$) and the public state (containing all the messages broadcast so far, denoted by ϕ_{public}). The role S is associated with a relation $\mathcal{R}(\mathsf{S})$ which basically specifies the correct behaviour of S as per the abstract protocol specifications. The functionality $\mathcal{F}_{\mathsf{VeSPa}}$ verifies this relation i.e. checks if the outgoing point-to-point and broadcast messages sent by S are computed correctly based on the incoming messages it received previously, the current public state and its private randomness (given as part of the witness). The messages that are verified are subsequently communicated. The formal description of $\mathcal{F}_{\mathsf{VeSPa}}$ appears below.

Functionality $\mathcal{F}_{\mathsf{VeSPa}}$ [KRY22]

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 the simulator $\mathcal{S}$. For corrupt roles R_i output x_i to the simulator $\mathcal{S}$. *Leak messages lengths and the broadcast message to the simulator in a rushing fashion.*

    If S is honest give $\textsc{Spoke}$ to S.
– On input $(\textsc{Read}, \mathsf{R}, \mathsf{S}, r')$ in round r where $r' < r$ for $x = y(r', \mathsf{S}, \mathsf{R})$ output x to R.
– On input $(\textsc{Read}, \mathsf{S}, r')$ in round r where $r' < r$ output $x = m(r', \mathsf{S})$ to R.

6.2 Extending to Verifiable State Propagation

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.

At a high-level, in order to emulate the invocation of $\mathcal{F}_{\mathsf{VeSPa}}$ by a role R in the abstract protocol, the machine assigned to execute role R does the following (1) first reads the bulletin board to obtain the broadcast messages and incoming point-to-point messages sent to R (by decrypting the relevant ciphertexts). (2) Then, according to the specifications of the underlying abstract protocol (i.e. as per the relation $\mathcal{R}(\mathsf{R})$ required by $\mathcal{F}_{\mathsf{VeSPa}}$ in the underlying protocol), it computes its outgoing point-to-point and broadcast messages based on the incoming messages and internal state. (3) prepares encryptions of these outgoing point-to-point messages using the encryption keys of the recipient roles. (4) Finally, the machine then invokes the $\mathcal{F}_{\mathsf{NIZK}}$ functionality with respect to a relation $\mathcal{R}_{\mathsf{VeSPa}}$ (described below) which essentially checks that the machine did the above actions (1), (2) and (3) correctly.

Accordingly, we define the 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 decryption is performed correctly.
– The incoming and outgoing plaintexts, and the public state satisfy the relation $\mathcal{R}(\mathsf{R})$ 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 $\mathsf{R.Rec}_{\mathsf{sid}}$, with elements of the form $(\mathsf{S}, \overline{x}_i)$, and past broadcast messages $\mathsf{Broadcast}_{\mathsf{sid}}$, with elements of the form (R, x), we define our relation,[8]

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

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 $\mathsf{Compile}(\Pi)$

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.

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

- Defines the statement $\phi \leftarrow (\mathsf{R}, \mathsf{sid}', \mathsf{R}.ek, \mathcal{R}_{\mathsf{sid}'}(\mathsf{R}), (\mathsf{R}_j.ek)_{j\in[k]}, \mathsf{R}.\mathsf{Rec}_{\mathsf{sid}'}, \mathsf{msg}, \mathsf{Broadcast}_{\mathsf{sid}'})$.
- Inputs $(\textsc{Verify}, \phi, \pi)$ to $\mathcal{F}_{\mathsf{NIZK}}$ with respect to the relation $\mathcal{R}_{\mathsf{VeSPa}}$. 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 $\mathsf{R}.\mathsf{Rec}_{\mathsf{sid}}$, if no such tuple exists $\perp$ should be output directly to the role. The ciphertext should then be decrypted to obtain $x_i \leftarrow \mathsf{PKE.Dec}(\mathsf{R}.dk, \overline{x}_i)$ 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 $\phi \leftarrow (\mathsf{R}, \mathsf{sid}', \mathsf{R}.ek, \mathcal{R}_{\mathsf{sid}'}(\mathsf{R}), (\mathsf{R}_j.ek)_{j\in[k]}, \mathsf{R}.\mathsf{Rec}_{\mathsf{sid}'}, \mathsf{msg}, \mathsf{Broadcast}_{\mathsf{sid}'})$ and witness $w \leftarrow (\mathsf{R}.dk, (x_j, \rho_j)_{j\in[k]}, w')$
- Inputs $(\textsc{Prove}, \phi, w)$ to $\mathcal{F}_{\mathsf{NIZK}}$ with respect to the relation $\mathcal{R}_{\mathsf{VeSPa}}$. 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.
- Post (msg, σ) to the bulletin board.
- Input $(\textsc{Ready})$ 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 in the $(\mathcal{F}_{\mathsf{VeSPa}}, \mathcal{F}_{\mathsf{BC\&SPP}})$-hybrid model, which is a compiler compatible secure implementation of $\mathcal{F}$ with threshold c/w (Definition 3). 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* $\epsilon > 0$*. Further, assume the schemes* PKE *and* SIG *used by* $\mathcal{F}_{\mathsf{RA}}$ *are IND-CPA and EUF-CMA secure respectively.*

Then, assuming a PKI setup, 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* $T < Nf_t$ *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* $\delta \geq 1$ *and* $0 < 1 - 2wf_t + wf_t^2$*.*[9]

The proof of Theorem 3 appears in the full version of this paper.

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.

[9] This holds when $f_t < 1 - \frac{\sqrt{w^2-w}}{w}$. For $w = 2$, namely when the abstract protocol withstands honest minority, this allows $f_t \approx 0.29$. For $w = 1.1$, namely when the abstract protocol withstands corruption of roughly 90% of the parties, this allows $f_t \approx 0.7$.

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. The committee size must not exceed some fixed size c, to ensure this we must fix the winning probability p such that the expected committee size is smaller than c allowing the application of a tail bound. To this end we let $p = c/((1+\epsilon')N)$ for some $\epsilon' > 0$.

Protocol Π_{RA}

Each machine M has access to a PKI containing KM-NCEpublic keys and VRF verification keys for each computation machine. VRF keys are generated by all machines invoking (malicious) key generation on $\mathcal{F}_{\mathsf{VRF}}$. Each machine additionally stores its current long-term KM-NCEsecret key as $M.sk$. 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.
- Input $(\textsc{EvalProve}, (\rho, \mathsf{cid}))$ to $\mathcal{F}_{\mathsf{VRF}}$ and wait for output $(\textsc{Evalutated}, \mathsf{draw}, \pi)$.
- If draw is a winning draw (i.e. $\mathsf{draw}/2^{\ell_{\mathsf{VRF}}} \leq p$), 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 $\mathsf{ctxt} \leftarrow KM - NCE.\mathsf{Enc}(M_{\mathsf{pid}'}.pk, (\mathsf{pid}', dk, sk))$.
- Erase the keys dk, sk and all randomness used for sampling the keys and pid', as well as any encryption randomness.
- Post $(\mathsf{cid}, ek, vk, \mathsf{ctxt}, \mathsf{draw}, \pi)$ to the bulletin board.

Read: On input $(\textsc{Read}, \mathsf{cid})$ 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, $(\mathsf{cid}, ek_1, vk_1, \mathsf{ctxt}_1, \mathsf{draw}_1, \pi_1), \ldots, (\mathsf{cid}, ek_k, vk_k, \mathsf{ctxt}_k, \mathsf{draw}_k, \pi_k)$.

3. Remove any elements $(\mathsf{cid}, ek_j, vk_j, \mathsf{ctxt}_j, \mathsf{draw}_j, \pi_j)$ posted by machine M from the list where draw_j is not a winning draw. This may be verified by inputting $(\textsc{Verify}, (\rho, \mathsf{cid}), \mathsf{draw}_j, \pi_j, M_{\mathsf{pid}}.vk^{\mathsf{VRF}})$ to $\mathcal{F}_{\mathsf{VRF}}$ where pid is the identifier of the machine which has posted the message to the bulletin board and ρ is the randomness the beacon has provided for committee cid. Remove the element if $\mathcal{F}_{\mathsf{VRF}}$ returns 0, or $\mathsf{draw}_j/2^{\ell_{\mathsf{VRF}}} > p$.
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 $(\mathsf{cid}, \perp, \perp, \perp)$.
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 $(\mathsf{cid}, ek_j, vk_j, \mathsf{ctxt}_j)$:
   - Append (ek_j, vk_j, C_j) to PKeys.
   - Attempt to decrypt $(\mathsf{pid}, dk, sk) \leftarrow KM - NCE.\mathsf{Dec}(M_j.sk, \mathsf{ctxt}_j)$. If $(\mathsf{pid}, dk, sk) \neq \perp$ 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 $(pk, sk, tk) \leftarrow KM - NCE.\mathsf{Gen}()$, 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. The proof of Theorem 4 appears in the full version of this paper.

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* $\mathsf{KM\text{-}NCE}$ *scheme used has* $\mathsf{KMNC_k\text{-}CCA}$ *(for* $\mathsf{k} = \mathsf{poly}(\kappa)$[10] *) and* $\mathsf{KM\text{-}NCE\text{-}UR}$ *security and the sortition has winning probability $c/((1+\epsilon')N)$ for $\epsilon' > 0$. Then, assuming a bare PKI setup, the protocol*

[10] 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.

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* $(\mathcal{F}_{\mathsf{Beacon}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{VRF}})$*-hybrid model.*

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 $\mathsf{YOSO} - \mathsf{ABB}$. 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.

Acknowledgements. Funded by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC), the Danish Independent Research Council under Grant-ID DFF-2064-00016B (YOSO), and the Digital Research Centre Denmark (DIREC).

References

[AHKP22] Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: SCALES - MPC with small clients and larger ephemeral servers. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022. Part II, volume 13748 of LNCS, pp. 502–531. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-22365-5_18

[BDO22] Braun, L., Damgård, I., Orlandi, C.: 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] Benhamouda, F., et al.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 260–290. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_10

[CCL+19] Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. Cryptology ePrint Archive, Report 2019/503 (2019). https://eprint.iacr.org/2019/503

[CDN01] Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_18

[CDPW07] Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_4

[CM19] Chen, J., Micali, S.: Algorand: a secure and efficient distributed ledger. Theoret. Comput. Sci. **777**, 155–183 (2019)

[DGKR18] David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 66–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_3

[DKI+23] David, B., Konring, A., Ishai, Y., Kushilevitz, E., Narayanan, V.: Perfect MPC over layered graphs. Cryptology ePrint Archive, Report 2023/330 (2023). https://eprint.iacr.org/2023/330

[EHK+13] Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_8

[GHK+21] Gentry, C., et al.: YOSO: you only speak once. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 64–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_3

[GHM+21] Gentry, C., Halevi, S., Magri, B., Nielsen, J.B., Yakoubov, S.: Random-index PIR and applications. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13044, pp. 32–61. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90456-2_2

[GOS12] Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. J. ACM (JACM) **59**(3), 1–35 (2012)

[HLH+22] Huang, Z., Lai, J., Han, S., Lyu, L., Weng, J.: Anonymous public key encryption under corruptions. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. Part III, volume 13793 of LNCS, pp. 423–453. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-22969-5_15

[KMTZ13] Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_27

[KRY22] Kolby, S., Ravi, D., Yakoubov, S.: Constant-round YOSO MPC without setup. Cryptology ePrint Archive, Paper 2022/187 (2022). https://eprint.iacr.org/2022/187

[MRV99] Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS, pp. 120–130. IEEE Computer Society Press, October 1999

[Nie02] Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: the non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_8