Asynchronous YOSO *a la* Paillier

Ivan Bjerre Damgård[1], Simon Holmgaard Kamp[2], Julian Loss[2], and Jesper Buus Nielsen[1]

[1] Aarhus University
[2] CISPA Helmholtz Center for Information Security

Abstract. We present the first complete asynchronous MPC protocols for the YOSO (You Speak Only Once) setting. Our protocols rely on threshold additively homomorphic Paillier encryption and are adaptively secure. We rely on the paradigm of Blum et al. (TCC '20) in order to recursively refresh the setup needed for running future steps of YOSO MPC, but replace any use of heavy primitives such as threshold fully homomorphic encryption in their protocol with more efficient alternatives. In order to obtain an efficient YOSO MPC protocol, we also revisit the consensus layer upon which our protocol is built. To this end, we present a novel total-order broadcast protocol with subquadratic communication complexity in the total number M of parties in the network and whose complexity is optimal in the message length. This improves on recent work of Banghale et al. (ASIACRYPT '22) by giving a simplified and more efficient broadcast extension protocol for the asynchronous setting that avoids the use of cryptographic accumulators.

Table of Contents

1 Introduction

The YOSO (You Speak Only Once) paradigm [GHK+21] is a recent promising paradigm for secure multiparty computation for a large set of M parties, where the work is done by a sequence of small committees, where each committee only speaks once, and is anonymous until it speaks to hide from an adaptive adversary. Because the committee size is much smaller than M, one can hope to have better complexity than conventional solutions for M players. We present several novel asynchronous protocols for the YOSO paradigm. Our protocols are the first to give a full-stack implementation of YOSO MPC in the fully asynchronous model that do not rely on heavy primitives such as threshold fully homomorphic encryption. The paper has two main contributions.

1. We present protocols for asynchronous YOSO role assignment (RA) and YOSO MPC using a recursive approach where YOSO RA is used to do YOSO MPC and YOSO MPC is used to do YOSO RA. This is is inspired by [BKLZL20]. While [BKLZL20] and [GHM+21] are based on non black-box use of FHE, we rely only on the security of Damgård-Jurik/Paillier encryption and so get better concrete efficiency. Our protocols are secure against adaptive corruptions.
2. The YOSO MPC and YOSO RA need a total-order broadcast channel. We also provide a novel YOSO asynchronous total-order broadcast protocol. The protocol can be seen as a YOSOfication of [KN23] using techniques from [BKLZL20,BLZLN22,DXR21]. It is message length optimal, meaning that there is no overhead associated with sending messages on the TOB if they are sufficiently large. In addition, the protocol is subquadratic in the total number M of parties in the network. While the latter could already be achieved using [BLZLN22] for messages that are polynomial in λ bits, we present a concretely efficient protocol.

Asynchronous YOSO from Paillier Encryption The YOSO MPC uses a version of CDN [CDN01] adapted to the asynchronous model. In CDN [CDN01] there is a public Paillier modulus N and the secret key is secret shared among a committee. As already noted in [GHK+21] this protocol can be made YOSO by having the secret key shared among a sequence of anonymous committees which will each handle the decryption of a single batch of ciphertexts. In [GHK+21] a synchronous static secure protocol using generic tools is sketched for this. We present an asynchronous adaptive secure YOSO construction and give concrete subprotocols and ZK proofs. We work in the random oracle model and assume we are given set-up data, such as the public key for the Paillier scheme. Comparing efficiency to previous work, we note that [BKLZL20] and [GHM+21] are based on threshold FHE and need to evaluate the entire key set-up of the threshold FHE inside FHE. In contrast, we make extensive use of the algebraic properties of our primitives and we do not need to convert any of them to circuits, indeed none of our zero-knowledge proofs are based on generic techniques.

Once we have an asynchronous MPC we follow the idea of [GHM+21] of doing YOSO RA by simulating a PIR client in MPC. As a main technical contribution we show how to do this with concrete efficiency: we first propose a novel 2-message PIR protocol based on Damgård-Jurik/Paillier encryption. It has overhead logarithmic in the size of the database,

and has the major advantage that its client part is very well suited for implementation in a CDN MPC based on Paillier, since the message from the server is effectively a Paillier encryption of the database item the client wants. With this machinery, we sample a random party to be member of a committee by considering set of public keys $\mathsf{gpk}_1, \ldots, \mathsf{gpk}_M$ of all parties as the database, retrieve a random one of these, randomize it inside MPC, and finally output the randomized key. This allows to send data to an anonymous committee and is the main part of what is known as role assignment (RA) in YOSO. We propose an El Gamal style cryptosystem for the gpk_i's that uses arithmetic modulo N, hence the randomization also uses aritmetic modulo N and so is efficient using our MPC engine. Further, encryption under gpk_i is secure, even given the factors of N which is important for our security proofs.

In our protocol, we need to generate many random encrypted bits. A classic trick for doing this efficiently is from [DFK+06], but relies on computing square roots modulo a prime. This cannot be done efficiently modulo N (unless we could factor), so we propose to use instead the Jacobi symbol of a random number modulo N, allowing us to get a solution with efficiency comparable to [DFK+06].

We achieve adaptive security using techniques similar to those used in [DN03] where a synchronous, adaptive secure, but non-YOSO CDN-style protocol was presented. A challenge, however, is that the UC simulator needs to be given a trapdoor allowing it to fake decryptions of ciphertexts that contain incorrect values, and the solution from [DN03] is incompatible with YOSO. We suggest a new solution exploiting that we have a random oracle.

A final contribution is that we show how to avoid that the size of the shares grow indefinitely as we reshare the secret global key by using Linear Integer Secret Sharing [DT06]. This was a problem in all previous YOSO protocols based on the CDN paradigm.

We need to make assumptions that go somewhat beyond what is standard for the primitives we use. First of all, it would be extremely inefficient to generate new key set-up for Paillier inside MPC. So, the modulus N we are given as set-up must be used for the entire protocol. Morerover, the YOSO framework requires players to delete their state once they have spoken, so we need to generate new keys for parties inside MPC, which means that the secret keys gsk_i will be encrypted under the Paillier public key. But on the other hand, the corresponding public keys gpk_i must be used for encrypting shares of the secret Paillier key when it is reshared for future committees. This creates circularity so we need to assume that our cryptosystems are secure despite this. We also need a selective opening type of assumption for the El-Gamal style encryptions which is essential towards getting adaptive security. For details, see Section 4.

Efficient Subquadratic Total Order Broadcast for the YOSO Model As part of our overall YOSO MPC protocol, we present a highly efficient total order broadcast (TOB) protocol with subquadratic communication complexity for adaptive corruptions in a hybrid model assuming a perfect coin flip functionality. We use our YOSO MPC to implement the coin-flip with subquadratic communication complexity. Our protocol relies on the [BKLZL20] paradigm of having an initial setup that allows flipping λ coins to run a TOB, and then using the TOB to recompute a new setup for future iterations. While [BKLZL20] is the first work to break

the n^2 communication barrier for binary byzantine agreement (BBA) in the asynchronous setting with adaptive corruptions, it is concretely quite heavy on communication and does not explain how to obtain an efficient TOB from BBA. We improve on the state of the art by taking the protocol for Agreement on a Core Set from [KN23] and adapting it to the YOSO framework. This allows us to get subquadratic communication with adaptive corruptions, by moving to the setting with suboptimal resilience against $T < (1-\epsilon)M/3$ corruptions studied in [BKLZL20,BLZLN22] where we can sample a fresh committee with honest supermajority for every round of the protocol. In this setting, we instantiate the RB protocol by [DXR21] for committees by using Reed-Solomon codes with reconstruction thresholds linear in n over a field of size at least M. This results in a subquadratic extension protocol to attain RB with optimal complexity in the message length, similar to the work of Banghale et al. [BLZLN22]. However, our protocol appears to be substantially simpler and more efficient, as it requires neither accumulators nor any application of advanced concentration bounds such as McDiarmid's inequality. Finally, we add some machinery to reach agreement on how much setup was consumed by the protocol. This allows the TOB to only consume expected constant number of RA committees per block, as opposed to the worst case λ.

2 Preliminaries

Paillier and Damgård-Jurik encryption Most of the material in this subsection is taken from [DJN10] where more details and proofs of the facts stated here can also be found. An exception is the El-Gamal inspired variant from a later subsection, which is a contribution of this paper.

We use several variants of the Paillier cryptosystem. It works with an RSA modulus $N = pq$ where the prime factors p, q are of form $p = 2p' + 1, q = 2q' + 1$ and p', q' are also primes.

We use the Damgård-Jurik generalization of Paillier, where the plaintext space is N^s for $s \geq 1$, and the basic form of an encryption of a message $m \in \mathbb{Z}_{N^s}$ with randomness $r \in \mathbb{Z}_N^*$ is $(N+1)^m r^{N^s} \bmod N^{s+1}$. This encryption scheme is the well-known to be additively homomorphic modulo N^s. Moreover, it is CPA-secure under the well-known Decisional Composite Residuosity Assumption (DCRA). It can be shown that this assumption for $s = 1$ implies the same assumption for any polynomial size s.

To simplify matters in the following, we will assume that the randomness for the encryption is always chosen to have Jacobi symbol +1. It can be shown that this variant is CPA secure under the same assumption.

A lossy version. We will make use of a variant where the element $(N + 1)$ is replaced by an encryption of 1, denoted by w_s, and determined as follows. We will consider values of s up some maximum value S, then we set $w_S = (N + 1)u_S^{N^S} \bmod N^{S+1}$, for random u_S, and $w_s = w_S \bmod N^{s+1}$. It is easy to see that all w_s are encryptions of 1. We will consider w_S as part of the public key. Using this notation, we define

$$E_{N,w_s}(m; r) = w_s^m r^{N^s} \bmod N^{s+1}$$

It is easy to see that $E_{N,w_s}(m;r)$ is actually an encryption of m according to the basic scheme above, it therefore also inherits the homomorphic property of the basic scheme. Concretely, we have for plaintexts m_1, m_2, a and randomness r_1, r_2 that

$$E_{N,w_s}(m_1;r_1)^a \cdot E_{N,w_s}(m_2;r_2) \bmod N^{s+1} = E_{N,w_s}(am_1 + m_2 \bmod N^s; r_1^a r_2 \bmod N)$$

The point of the variant using w_s is that it is a possibly lossy encryption scheme. Namely, if we replace w_S by a random encryption of 0 $\bar{w}_S$, then an "encryption" of any m, $E_{N,\bar{w}_s}(m;r) = \bar{w}_s^m r^{N^s} \bmod N^{s+1}$ will actually be an encryption of 0. However, given only the public key, the two forms cannot be distinguished, as an encryption of 0 is indistinguishable from an encryption of 1.

How to decrypt For decryption, we define a decryption exponent d_S constructed by the Chinese remainder theorem such that $d_S \equiv 0 \bmod \phi(N)$ and $d_S \equiv 1 \bmod N^S$. We can use d_S to decrypt any $E_{N,w_s}(m;r)$ where $s \leq S$, as we have:

$$E_{N,w_s}(m;r)^{d_S} = (N+1)^{md_S}((ru_s)^{N^s})^{d_S} \bmod N^{s+1} = (N+1)^m \bmod N^{s+1}.$$

We can then get the message by exploiting the fact that discrete logs modulo $(N+1)$ are easy to compute. Later, we will define a threshold version of the cryptosystem where d_S is shared among a set of parties, who can then collaborate to raise a ciphertext to power d_S, thus effectively decrypting it. In the following, we will sometimes suppress the randomness from the notation for readability, and write $E_{N,w_s}(m)$ instead of $E_{N,w_s}(m;r)$.

"El Gamal" style encryption and signatures A final variant we will need is a scheme that allows several parties to use the same modulus N, we will be able to do both encryption and signatures and neither decryption nor signing will require the factorization of N. We first fix an element $g \in \mathbb{Z}_N^*$ of order $p'q'$. A party P_i will have a secret key $\mathsf{gsk}_i = (x_i, \tilde{x}_i)$, both components chosen at random in $\mathbb{Z}_N$, while the public key will be $\mathsf{gpk}_i = (g, h_i, \tilde{h}_i) = (g, g^{x_i} \bmod N, g^{\tilde{x}_i} \bmod N)$ [3]. Here, x_i will be used for decryption while $\tilde{x}_i$ will be used for signatures.

Encryption We define $E_{\mathsf{gpk}_i}(m;r) = (g^r \bmod N, g^m h_i^r \bmod N)$. Here, m must be small enough that taking discrete log modulo g is feasible, and assuming this, decryption given x_i is straightforward. In the following, we will need to encrypt numbers m that are too large for discrete log to be feasible. In such cases, we will split m in chuncks $m_1, \ldots, m_a$ of size γ bits, for a small enough γ: $m = \sum_{j=0}^{a} 2^{\gamma j} m_j$. We then encrypt each m_j individually. We will nevertheless write this as $E_{\mathsf{gpk}_i}(m;r)$. Looking ahead, the fact that the m_i are in the exponent and that there is a linear relationship between the m_j's and m, allows for efficient zero-knowledge proofs for these encryptions.

A very important observation is that, using the Chinese remainder theorem, one can easily show that if DDH is a hard problem in the group of squares modulo both p and

[3] Since N is, relatively speaking, very close to $\phi(N)$, choosing x_i as described will result in h_i being statistically close to a random element in the group generated by g.

q, then this cryptosystem is secure, *even if the factorization of N is known.* Note that in [DJN10] a different "El Gamal style" variant of Paillier was proposed. However, that variant is insecure if the factorization of N is known.

Signatures. A party P_i will be able to sign messages using $\tilde{x}_i$ and the corresponding public $\tilde{h}_i$. For this, we will use a standard Σ-protocol for proving knowledge of $\tilde{x}_i$, the discrete log base g of $\tilde{h}_i$. This protocol can then be transformed to a signature scheme in the random oracle model following the Fiat-Shamir paradigm. We denote a signature on message m by $\mathsf{Sig}_{\mathsf{gsk}_i}(m)$ and the corresponding verification of signature σ on m by $\mathsf{Ver}_{\mathsf{gpk}_i}(\sigma, m)$. With a standard type of arguments this scheme can be shown CMA-secure if the discrete log problem is hard in $\mathbb{Z}_N^*$. It can be seen as variant of the well-known Schnorr signature scheme and we leave the details to the reader. However, we again observe that even if the factorization of N is known, the scheme remains secure, assuming discrete log is hard in $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$, which can be seen using the Chinese Remainder theorem.

Randomized Keys. In the following we will be using also randomized versions of the public keys, of form $(g^s \bmod N, h_i^s \bmod N, g^{s'} \bmod N, \tilde{h}_i^{s'} \bmod N)$ for random s, s'. Note that the owner of the original key pair also knows the discrete log base g^s of h_i^s, and of $g^{s'}$ of $\tilde{h}_i^{s'}$ namely $x_i, \tilde{x}_i$ and so can identify a randomized key as coming from his key, can decrypt messages encrypted under the randomized key, and make signatures that can be verified under the randomized key.

Commitment Scheme We will need a commitment scheme allowing commitment to integers. We will use the scheme suggested by Fujisaki and Okamoto [FO99], for a formal treatment, see [DF02]. Here, we describe the properties we need informally. The scheme is based on a modulus N' of the same form as our Paillier modulus, but chosen independently. The public key for the scheme is now $\mathsf{ck} = (N, \alpha, \beta)$ where α, β are random squares mod N', and we assume ck is given as set-up. A commitment to integer x using randomness r is denoted $\mathsf{Com}_{\mathsf{ck}}(x; r) = \alpha^x \beta^r \bmod N'$. Here, x is an integer in an interval $[0..2^b]$, and r is chosen uniformly from an interval $[0..2^K]$. We return to the choice of K below. The commitment scheme is clearly homomorphic over both inputs to the commitment function, that is, we have

$$\mathsf{Com}_{\mathsf{ck}}(x; r) \cdot \mathsf{Com}_{\mathsf{ck}}(x'; r') = \mathsf{Com}_{\mathsf{ck}}(x + x'; r + r')$$

It is well known that this is statistically hiding and computationally binding.

Non-Interactive Zero-Knowledge Proofs In several cases, we will require non-interactive zero-knowledge proofs of knowledge. They allow a prover to prove, for a predicate P and a given (public) value x that they know a witness w such that $P(x, w) = 1$. We will denote such a proof by $\mathsf{NIZK}(w : P(x, w) = 1)$. These proofs need to be unconditionally simulation sound, statistical zero-knowledge and on-line extractable, i.e., there is an efficient extractor that can extract a witness from a successful prover without rewinding. In all cases, we can achieve this in the random oracle model: we first construct a Σ-protocol for the relation in question using standard techniques, and then we apply the Fishlin transformation to

get non-interactive proofs. More details can be found in Appendix G. With one exception, all Σ-protocols introduce only a constant factor overhead in communication which means that after the Fischlin transformation, we get an overhead factor of $\lambda/\log\lambda$, where λ is the security parameter. The exception is the protocol for so-called 2-level ciphertexts from Appendix E, which is a double discrete log type protocol that inherently seems to require binary challenges and therefore gives us an overhead factor λ.

Protocol PIR, to be executed by a client C and a server S.
S has a database consisting of numbers in $\mathbb{Z}_N$, $y_0, \ldots, y_{L-1}$, and we assume for simplicity that L is a two-power, $L = 2^\lambda$. C wants to retrieve y_t for some $0 \leq t \leq L$.

1. C writes t in binary as $t_0, t_1, ..., t_{\lambda-1}$ where t_0 is the *most* significant bit. C sends
$$c_0 = E_{N,w_1}(t_0), \ldots, c_{\lambda-1} = E_{N,w_\lambda}(t_{\lambda-1})$$
to S.
2. S sets $(y_0^0, ..., y_{L_1}^0) = (y_0, ..., y_{L-1})$, and now executes the following loop for $i = 0$ to $\lambda - 1$:
   (a) Set $\bar{c}_i = E_{N,w_i}(1) \cdot c_i^{-1} \bmod N^{i+1}$, where $E_{N,w_i}(1)$ is an encryption of 1 with default randomness 1. Note that $\bar{c}_i$ is an encryption of the bit $1 - t_i$.
   (b) Define $L_i = L/2^i$. This step takes as input the list $(y_0^i, ..., y_{L_i-1}^i)$, and outputs a list $(y_0^{i+1}, ..., y_{L_i/2-1}^{i+1})$. Namely, for $j = 0$ to $L_i/2 - 1$, set
$$y_j^{i+1} = c_i^{y_j^i} \cdot \bar{c}_i^{y_{j+L_i/2}^i} \bmod N^{i+1}.$$
   Note that y_j^{i+1} is an encryption of $t_i \cdot y_j^i + (1 - t_i) \cdot y_{j+L/2^i}^i$, and hence the loop produces encryptions of the first or the second half of the incoming list, depending on t_i.
3. The list output by the last loop step above has lenght $L_\lambda = L/2^\lambda = 1$. S sends the single ciphertext on the list y_0^λ to C.
4. Note that we have
$$y_0^\lambda = E_{N,w_{\lambda-1}}(E_{N,w_{\lambda-2}}(\ldots E_{N,w_1}(y_t))),$$
so by doing $\lambda - 1$ decryption steps, C can retrieve y_t, as desired.

Fig. 1. The PIR protocol.

3 A PIR protocol based on Damgård-Jurik encryption

In this section, we present a PIR protocol (Fig. 1), in the standard form with a single client and server, where we assume that the client has the secret key for an instance of the Damgård-Jurik scheme as defined above. In a later section we show how to transplant it to a multiparty setting, where several parties play the role of both server and client and we will see how this can be used to do role assignment.

A first important observation that will come in handy is the following: the plaintext space of the encryption function $E_{N,w_s}(\cdot;\cdot)$ is $\mathbb{Z}_N^s$, but this is also the ciphertext space of $E_{N,w_{s-1}}(\cdot;\cdot)$. Therefore, given ciphertexts $E_{N,w_{s-1}}(m)$, $E_{N,w_s}(m')$ and the public key, one can compute $E_{N,w_s}(m')^{E_{N,w_{s-1}}(m)} \bmod N^{s+1} = E_{N,w_s}(m' \cdot E_{N,w_{s-1}}(m) \bmod N^s)$.

Game $\mathsf{SOINDCPA}^b_{\mathsf{PEAS},\mathsf{PPBox},A}$

Init: Initially generate $(\mathsf{N},\mathsf{sp}) \leftarrow \mathsf{Gen}$ and give (N,sp) to A. Let $j = r = 0$.
IND-CPA: On $(\textsc{EncRole}, s^0, s^1)$ from A initialise $\mathsf{PPBox}(s^b)$.
Generate Committee: On input $(\textsc{NextCommittee})$ let $j \leftarrow j+1$ and do:
1. For $i = 1,\ldots,n$ sample $(\mathsf{gpk}_i^j, \mathsf{gsk}_i^j) \leftarrow \overline{\mathsf{Gen}}$, $\mathsf{rpk}_i^j \leftarrow \mathsf{Ran}(\mathsf{gpk}_i^j)$.
2. Input $((\mathsf{gpk}_1^j, \mathsf{rpk}_1^j), \ldots, (\mathsf{gpk}_n^j, \mathsf{rpk}_n^j))$ to A.
3. Let $L^j = \emptyset$ be the set of global secret keys leaked so far.

Leak Global Secret Key: On input $(\textsc{SecretKey}, \mathsf{gpk}_i^j)$ where $L^j \cup \{i\} \in \mathcal{A}$, input the corresponding gsk_i^j to A and let $L^j = L^j \cup \{i\}$.
Run PPBox: On input $(\mathrm{PPBox}, \mathsf{aux}, j)$ let $r \leftarrow r+1$ and do:
1. Input aux to PPBox.
2. Sample $(\mathsf{aux}', \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{PPBox}$.
3. For $i = 1,\ldots,n$ let $c_i \leftarrow \overline{\mathsf{Enc}}_{\mathsf{rpk}_i^j}(\mathsf{sh}_i)$.
4. Input $(\mathsf{aux}, c_1, \ldots, c_n)$ to A.

Guess: The adversary ends the game by outputting a guess $g \in \{0,1\}$.

Fig. 2. The Selective Opening IND-CPA Game for Role Encryption.

The PIR protocol is clearly secure based on CPA security: S cannot distinguish C's message from a set of random encryptions, and if S is semi-honest, C will always get correct output. The protocol requires only 2 messages which is clearly optimal.

Note that we can allow the input y_i's to be longer than the modulus, say numbers in $\mathbb{Z}_{N^v}$, we just need to define C's message to be

$$c_0 = E_{N,w_v}(t_0), \ldots, c_{\lambda-1} = E_{N,w_{v+\lambda}}(t_{\lambda-1}),$$

and adjust S's part accordingly. If we denote the length of a data item by k and the length of N by κ, one finds that the communication complexity is $O(\lambda k + \lambda^2\kappa)$. Thus, for large k, the overhead over just sending a data item in the clear is only a factor λ, logarithmic in the size of the database.

4 Syntax and Security Notions for Encryption and Signature

For later use we make some syntax capturing the above Paillier-based encryption and authentication system PEAS. We have a tuple of algorithms $(\mathsf{Gen}, \overline{\mathsf{Gen}}, \mathsf{Enc}, \mathsf{Dec}, \overline{\mathsf{Enc}}, \overline{\mathsf{Dec}}, \mathsf{Ran}, \mathsf{Sig}, \mathsf{Ver})$, where $(N, \mathsf{sp}) \leftarrow \mathsf{Gen}$ is base key generator generator, N the public parameter, $\mathsf{sp} = (p,q)$ the secret parameters, $c \leftarrow \mathsf{Enc}_{N^s}(m;r)$ is base encryption with ciphertext space N^s, $m \leftarrow \mathsf{Dec}_{\mathsf{sp}}(c)$ is base decryption, $(\mathsf{gpk},\mathsf{gsk}) \leftarrow \overline{\mathsf{Gen}}(\mathsf{N})$ the derived key generator for the ElGamal variant, which given the public parameters generate the an encryption key gpk and a decryption key gsk, $c \leftarrow \overline{\mathsf{Enc}}_{\mathsf{gpk}}(m;r)$ is encryption, $m \leftarrow \overline{\mathsf{Dec}}_{\mathsf{gsk}}(c)$ is decryption, and Ran can be used to rerandomize an encryption key: if $(\mathsf{gpk},\mathsf{gsk}) \leftarrow \overline{\mathsf{Gen}}(\mathsf{N})$ and $\mathsf{rpk} \leftarrow \mathsf{Ran}(\mathsf{gpk})$ then $(\mathsf{rpk},\mathsf{gsk}) \in \overline{\mathsf{Gen}}(\mathsf{N})$. Furthermore, given gsk one can in PPT recognise rpk as a rerandomisation of gpk. One can sign a message as $\sigma \leftarrow \mathsf{Sig}_{\mathsf{gsk}}(m)$ and it can be verified as $\mathsf{Ver}_{\mathsf{gpk}}(m,\sigma)$ and also as $\mathsf{Ver}_{\mathsf{rpk}}(m,\sigma)$. We need that the signature scheme is existentially unforgeable un-

der adaptive chosen message attack (EUF-CMA) even against an adversary being given sp. This is a standard notion and we do not recall it here.

Game $\mathsf{SOANON}_{\mathsf{PEAS},A}$

Init: Initially generate $(\mathsf{N},\mathsf{sp}) \leftarrow \mathsf{Gen}$ and give (N,sp) to A.
Generate Permuted Keys: After initialisation:

1. Let $\pi : G \to R$ be a uniformly random permutation.
2. For $i = 1,\ldots,\ell$ sample $(\mathsf{gpk}_i^*, \mathsf{gsk}_i^*) \leftarrow \overline{\mathsf{Gen}}$.
3. For $i = 1,\ldots,\ell$ sample $\mathsf{rpk}_i^* \leftarrow \mathsf{Ran}(\mathsf{gpk}_{\pi^{-1}(i)}^*)$.
4. Let $G = R = [\ell]$ be the global keys resp. role keys not leaked so far.
5. Input $((\mathsf{gpk}_1^*, \mathsf{rpk}_1^*),\ldots,(\mathsf{gpk}_n^*, \mathsf{rpk}_n^*))$ to A.

Leak Global Secret Key: On output $(\textsc{SecretKey}, \mathsf{gpk}_i^*)$ from A input gsk_i^* to A and let $G = G \cup \{i\}$ and $R = R \cup \{\pi(i)\}$.
Guess: The adversary ends the game by outputting $(i,j) \in G \times R$. Let $g \in \{0,1\}$ be 1 iff $\pi(i) = j$. The output of the game is the number $x = g - 1/|R|$.

Fig. 3. The Selective Opening Anonymity Game.

We need that $\overline{\mathsf{Enc}}$ is IND-CPA even against an adversary being given sp. We argued this in Section 2, but we need it to hold under a selective opening attack. We formulate it via a secret sharing: if the adversary selectively open encryptions of shares not in the access structure we assume the remaining encryptions are IND-CPA. We consider a generalisation of secret sharing which we call a privacy preserving box. A PPBox PPBox takes as input a secret s. The adversary can then repeatedly input an auxiliary input aux to PPBox making it generate $(\mathsf{aux}', \mathsf{sh}_1,\ldots,\mathsf{sh}_n)$, where aux' is an auxiliary output and the sh_i's are shares meant for a set of n parties. The adversary may corrupt parties P_i. In response to this it gets all the shares sh_i of P_i, future and past. We say that Q is in the privacy structure of PPBox if seeing all shares sh_i for $i \in Q$ gives a view statistically independent of s. We call the set of such Q the privacy structure $\mathcal{A}$. We will later use a concrete instance of PPBox to model all the sharings and resharings of sp that happen in our protocol. We note that our notion of is related to what is normally called *weak* SO, but seems even weaker because we do not formulate the notion via message resampling. In particular the impossibility result for strong SO in [ORV14] does not apply. It is plausible that techniques in [FHKP16] can be used to prove SO-IND-CPA for our ElGamal-based scheme, but we leave this as future work.

Definition 1 (Selective Opening IND-CPA for Role Encryption). *Let* PPBox *be a PPBox with privacy structure* $\mathcal{A}$*. We say that* $(\mathsf{PEAS},\mathsf{PPBox})$ *is selective opening IND-CPA (SO-IND-CPA) secure if for all PPT* A *it holds that* $\mathsf{SOINDCPA}^0_{\mathsf{PEAS},\mathsf{Share},A} \approx \mathsf{SOINDCPA}^1_{\mathsf{PEAS},\mathsf{Share},A}$.

We also need that it is hard to connect a role key $\mathsf{rpk} \leftarrow \mathsf{Ran}(\mathsf{gpk})$ to the global key gpk if one does not know gsk. This holds under the DDH assumption (Section 2), but again we need it under selective opening. We make the assumption that this is the case. The game is in Fig. 3. The adversary can always make a random guess $(i,j) \in G \times R$. In this case

Game $\mathsf{CSOINDCPA}^b_{\mathsf{PEAS},\mathsf{PPBox},A}$

Init: Initially generate $(\mathsf{N},\mathsf{sp}) \leftarrow \mathsf{Gen}$ and give N to A. Let $r = 0$. Let $\mathcal{A}$ be the privacy structure of PPBox. Initialize $\mathsf{PPBox}(\mathsf{N},\mathsf{sp})$.

Generate Committee: On input (NextCommittee) let $j \leftarrow j+1$ and do:

1. For $i = 1,\ldots,n$ sample $(\mathsf{gpk}_i^j, \mathsf{gsk}_i^j) \leftarrow \overline{\mathsf{Gen}}$, $\mathsf{rpk}_i^j \leftarrow \mathsf{Ran}(\mathsf{gpk}_i^j)$.
   - Call the random tape used to generate all keys r_j.
2. Input $((\mathsf{gpk}_1^j, \mathsf{rpk}_1^j),\ldots,(\mathsf{gpk}_n^j, \mathsf{rpk}_n^j))$ to A.
3. Let $L^j = \emptyset$ be the set of global secret keys leaked so far.

Leak Global Secret Key: On input $(\textsc{SecretKey}, \mathsf{gpk}_i^j)$ where $L^j \cup \{i\} \in \mathcal{A}$, input gsk_i^j to A and let $L^j = L^j \cup \{i\}$.

Run PPBox: On input $(\textsc{PPBox}, \mathsf{aux}, j)$ let $r \leftarrow r+1$ and do:

1. Input aux to PPBox.
2. Sample $(\mathsf{aux}', \mathsf{sh}_1,\ldots,\mathsf{sh}_n) \leftarrow \mathsf{PPBox}$.
3. For $i = 1,\ldots,n$ let $c_i \leftarrow \overline{\mathsf{Enc}}_{\mathsf{rpk}_i^j}(\mathsf{sh}_i)$.
4. Input $(\mathsf{aux}, c_1,\ldots,c_n)$ to A.

Base IND-CPA: On input $(\textsc{EncBase}, f : \{0,1\}^* \to \mathbb{Z}_N^a)$ do:

1. Compute $(m_1^1,\ldots,m_a^1) = f(\mathsf{N}, r_1,\ldots,r_j)$.
2. Let $(m_1^0,\ldots,m_a^0) = (0,\ldots,0)$.
3. Input $(c_1,\ldots,c_a) \leftarrow (\mathsf{Enc}_\mathsf{N}(m_1^b),\ldots,\mathsf{Enc}_\mathsf{N}(m_a^b))$ to A.

Guess: The adversary A ends the game by outputting a guess $g \in \{0,1\}$.

Fig. 4. The Circular Selective Opening IND-CPA Game

$\mathrm{E}[g] = 1/|R|$ and $\mathrm{E}[x] = 0$, so $\mathrm{E}[x]$ is how much better than random the adversary can guess an unrevealed role key.

Definition 2 (Selective Opening Anonymity). *We say that* PEAS *is SO-ANON secure if for all PPT* A *it holds that* $\mathrm{E}[\mathsf{SOANON}_{\mathsf{PEAS},A}] = \mathrm{negl}(\lambda)$.

We finally need that the base system of PEAS has a strong notion of circular security under selective opening, where it is secure even if encryptions of secret sharings of the secret parameters sp are given to the adversary under role keys and representations of the corresponding secret keys are encrypted under N.

Definition 3 (Circular Selective Opening IND-CPA). *Let* PPBox *be a PPBox with privacy structure* $\mathcal{A}$*. We say that* $(\mathsf{PEAS}, \mathsf{PPBox})$ *is CSO-IND-CPA secure if for all PPT* A *it holds that* $\mathsf{CSOINDCPA}^0_{\mathsf{PEAS},\mathsf{PPBox},A} \approx \mathsf{CSOINDCPA}^1_{\mathsf{PEAS},\mathsf{PPBox},A}$.

5 UC Modelling

We use the UC framework in [CCL15] as it is asynchronous, allows to model interactive functionality and is simple and sufficient for our study, as we work with a fixed set of parties $\mathbb{P} = \{P_1,\ldots,P_M\}$. We model functionalities for total-order broadcast ($\mathcal{F}_{\mathrm{TOB}}$) and YOSO role assignment with MPC and coin-flip ($\mathcal{F}_{\mathrm{RA+MPC+CF}}$). We also have a helper $\mathcal{F}_{\mathrm{SETUP}}$ for setting up values. As the underlying model of communication we assume authenticated point-to-point communication with atomic send, where a party can in one instruction can send

messages to several users, and it is guaranteed that all messages be delivered even if the party gets corrupted right after the sending. See [BKLZL20] for further discussion. For completeness an ideal functionality is given in Fig. 28. When formulating ideal functionalities we talk about the adversary having to eventually deliver certain values to guarantee liveness. We discuss in Appendix H.1 how this is formalized, but the exact formulation is not consequential for the security of our protocols.

Functionality $\mathcal{F}_{\text{TOB}}$

Init: Let $L = \epsilon$ be the empty ledger. For each party P let $\ell_P = 0$ be the number of blocks delivered at P. Let $b = 0$ be the number of blocks produced so far. Let b_P be the number of blocks requested by P.

Broadcast: On input $(\text{BROADCAST}, \mathsf{mid}, m)$ from party $P(\mathsf{mid})$ add (mid, m) to $\mathsf{Accepted}$ and leak $(\text{BROADCAST}, \mathsf{mid}, m)$ to the adversary.

Set Wait Predicate: On input (WAIT, W) from honest party P let $b_P \leftarrow b_P + 1$ and let $W^{b_P} = W$. We require that all honest parties agree on W^{b_P}.

Next Batch: On input $B = ((\mathsf{mid}_1, m_1), \ldots, (\mathsf{mid}_\ell, m_\ell))$ from the adversary where W^{b+1} is defined and where $(\mathsf{mid}_j, m_j) \in \mathsf{Accepted}$ for $j = 1, \ldots, \ell$ and $W^{b+1}(L, B) = \top$, update $L \leftarrow L \| B$ and remove each (mid_j, m_j) from $\mathsf{Accepted}$. Update $b \leftarrow b + 1$.

Deliver: On input $(\text{DELIVER}, P)$ from the adversary where $\ell_P < |L|$ update $\ell_P = \ell_P + 1$ and output $L[\ell_P]$ to P.

Eventual Liveness: If $b_P > b$ for all honest parties and $W^{b+1}(L, \mathsf{Accepted}) = \top$ then eventually the adversary must produce block number $b + 1$. Furthermore, if $(\mathsf{mid}, m) \in \mathsf{Accepted}$ and $P(\mathsf{mid})$ is honest, then eventually (mid, m) must be added to a block. Finally, if $\ell_P < |L|$ then eventually the adversary will deliver a new message to P.

Fig. 5. Total-Order Broadcast

Total Order Broadcast Our ideal functionalities will have a notion of batches. The reason is that our implementation of total-order broadcast will use keys for YOSO role assignment and will also be used to produce such keys. It is therefore important that the total-order broadcast is not wasting all its setup on sending dummy messages by the corrupted parties. It should consume setup only when it can add messages to the ledger which will help make progress on producing fresh RA keys. We introduce a notion of a wait predicate W which tells the ideal functionality when it can produce the next batch. This is formulated via a notion of blocks of messages and such a block being a valid extension of the current ledger. A block B is a non-empty sequence of bit-strings B and a ledger L is a sequence of blocks. A wait predicate $W(L, B) \in \{\top, \bot\}$ judges if B may extend L. We require that if $W(L, B) = \top$ then $W(L, B \| B') = \top$ for all blocks B' and $W(L, \pi(B)) = \top$ for all permutations of the messages in B. Messages are named by a message identifier mid. We assume that mid contains the name of the party allowed to send it, and we denote this party by $P(\mathsf{mid})$. We assume that $P(\mathsf{mid})$ uses mid at most once. The ideal functionality is given in Fig. 5. The formalisation is straightforward and uses known design patterns. For completeness there is a motivation in Appendix H.3

Functionality $\mathcal{F}_{\text{RA+MPC+CF}}^{\mathsf{PEAS},\epsilon,\mathbb{F},\gamma,1}$ for role assignment, MPC. Parametrised by an exclusion factor ϵ, a class $\mathbb{F}$ of feasible function and the number of coin-flips γ.

Init: When activated the first time do the following.

1. Generate $(\mathsf{N}, \mathsf{sp}) \leftarrow \mathsf{PEAS.Gen}$ and output N to all parties. Leak sp to the adversary.
2. Let $\mathsf{b} = 0$ be the number of batches produced.
3. For $P \in \mathbb{P}$ let $\mathsf{bo}_P = 0$ and $\mathsf{bd}_P = 0$ be the number of batches ordered by respectively delivered at P.

Order Next Batch: On input $(\textsc{NextBatch}, f \in \mathbb{F}_{\mathsf{N}}, W, x_P \in \mathbb{Z}_{\mathsf{N}})$ from honest party P where $\mathsf{bo}_P = \mathsf{bd}_P$ do the following:

1. Leak $(\textsc{NextBatch}, f, W, P)$ to the adversary.
2. Update $\mathsf{bo}_P \leftarrow \mathsf{bo}_P + 1$.
3. Let $f^{\mathsf{bo}_P} = f$, $W^{\mathsf{bo}_P} = W$, and $x_P^{\mathsf{bo}_P} = x_P$. We assume that honest parties agree on f^b and W^b for a given b.
4. For $j = 1, \ldots, \gamma$ sample uniformly random $\mathsf{coin}_j^{\mathsf{b}} \in \{0,1\}^\lambda$.

Generate Batch: We call $\mathbb{Q}$ qualified for batch b if $\mathsf{bo}_P \geq b$ for all honest $P \in \mathbb{Q}$ and $W^b(\mathbb{Q}) = \top$. On input $(\textsc{NextBatch}, \mathbb{Q} \subseteq \mathbb{P}, \mathbb{R} \subseteq \mathbb{Q}, \{x_P^{\mathsf{b}+1}\}_{P \in \mathbb{R}})$ from the adversary, where $\mathbb{Q}$ is qualified for batch $\mathsf{b}+1$ and $\mathbb{R}$ is the set of corrupted parties in $\mathbb{Q}$, do:

1. Update $\mathsf{b} \leftarrow \mathsf{b} + 1$.
2. Let $\mathbb{P}^{\mathsf{b}} = \mathbb{Q}$.
3. For all $P \in \mathbb{P}^{\mathsf{b}}$ generate $(\mathsf{gpk}_P^{\mathsf{b}}, \mathsf{gsk}_P^{\mathsf{b}}) \leftarrow \mathsf{PEAS.}\overline{\mathsf{Gen}}(\mathsf{N})$.
4. Let $\mathsf{gpk}^{\mathsf{b}} = \left\{\left(P, \mathsf{gpk}_P^{\mathsf{b}}\right)\right\}_{P \in \mathbb{P}^{\mathsf{b}}}$.
5. Let π be a uniformly random permutation of $\mathbb{P}^{\mathsf{b}}$, and let $R_P = \pi(P)$ be the role of $P \in \mathbb{P}^{\mathsf{b}}$.
6. For all $P \in \mathbb{P}^{\mathsf{b}}$ generate $\mathsf{rpk}_{R_P}^{\mathsf{b}} \leftarrow \mathsf{Ran}(\mathsf{gpk}_P^{\mathsf{b}})$.
7. Let $\mathsf{rpk}^{\mathsf{b}}$ be $\{(R, \mathsf{rpk}_R^{\mathsf{b}})\}_{R \in \mathbb{P}^{\mathsf{b}}}$.
8. Sample $(y^{\mathsf{b}}, \{(P, y_P^{\mathsf{b}})\}_{P \in \mathbb{P}^{\mathsf{b}}}) \leftarrow f^{\mathsf{b}}\left(\{(P, x_P^{\mathsf{b}})\}_{P \in \mathbb{P}^{\mathsf{b}}}\right)$.
9. Give $Y^{\mathsf{b}} = (\mathsf{gpk}^{\mathsf{b}}, \mathsf{rpk}^{\mathsf{b}}, y^{\mathsf{b}})$ to the adversary along with $\left(\mathsf{gsk}_P^{\mathsf{b}}, y_{\pi(P)}^{\mathsf{b}}\right)$ for all $P \in \mathbb{R}^{\mathsf{b}}$.

Deliver Batch: On input $(\textsc{DeliverBatch}, P)$ from the adversary, where $\mathsf{bd}_P < \mathsf{b}$, update $\mathsf{bd}_P = \mathsf{bd}_P + 1$ and output Y^{bd_P} to P. If $P \in \mathbb{P}^{\mathsf{bd}_P}$, then output $\left(\mathsf{gsk}_P^{\mathsf{bd}_P}, y_{\pi(P)}^{\mathsf{bd}_P}\right)$ to P.

Flip Coin: On input $(\textsc{Flip}, b, j)$ from P where $b \leq \mathsf{bd}_P$ and $j \in [\gamma]$ record $(\textsc{Flip}, P, b, j)$ and output coin_j^b to the adversary.

Deliver Coin: On input $(\textsc{DeliverCoin}, P, b, j)$ from the adversary, where $b \leq \mathsf{b}$ and $j \in [\gamma]$ output $(\textsc{Flip}, b, j, \mathsf{coin}_j^b)$ to P.

Eventual Liveness: If $\mathsf{bo}_P \geq \mathsf{b}$ for all honest $P \in \mathbb{P}$, then the adversary must eventually input a valid $(\textsc{NextBatch}, \cdot, \cdots)$. If $\mathsf{bd}_P < \mathsf{b}$ for some honest P then the adversary must eventually input $(\textsc{DeliverBatch}, P)$. If for some (b, j) the value $(\textsc{Flip}, Q, b, j)$ is recorded for all honest $Q \in \mathbb{P}$ and P is honest, then the adversary must eventually input $(\textsc{DeliverCoin}, P, b, j)$.

Corruption: On corruption of P output $\left(\mathsf{gsk}_P^b, y_{\pi(P)}^{\mathsf{bd}_P}\right)$ to the adversary iff $b \leq \mathsf{b}$ and $b > \mathsf{bd}_P$.

Fig. 6. The Ideal Functionality for Role Assignment, MPC, and Coin-Flip

Asynchronous Role Assignment The functionality for role assignment generates in each batch a fresh set of global public keys $(\mathsf{gpk}_1, \ldots, \mathsf{gpk}_M)$ one for each of the M parties. It then generates from each gpk_i a randomized rpk_i and outputs these in permuted order to hide which party has which role. Still, other parties can use rpk to encrypt messages for P_i. The party P_i will learn the secret key gsk_i of gsk_i and can use it to identify and decrypt encryptions under rpk_i. One modification to the above discussion is that not all M parties get new keys. We allow that the adversary can exclude a set of parties of size ϵM. We will later implement $\mathcal{F}_{\text{RA+MPC+CF}}$ with exclusion faction about $1/3$. The reason is that in the implementation we need to hear from a party before we can give it a new key. And in an asynchronous protocol we cannot hope to hear from all parties. The reason we need to hear from a party is that we want to send encrypted date to the party and we have adaptive corruption, so we need non-committing encryption. At the same time the receiving parties when acting an anonymous role is only allowed to speak once, so we need non-interactive, non-committing encryption, which has the drawback that the secret key grows linear in the amount of data which can be sent under a fixed public key [Nie02]. So we need that parties occasionally refreshes their public keys. We opted for a solution establishing the non-committing encryption channel between the MPC and P. When generating a new batch P will pick a fresh one-time pad $\mathsf{otp}_P^{\mathsf{b}}$ and send $E_{\mathsf{N}}(\mathsf{otp}_P^{\mathsf{b}})$ on the TOB. It then deletes all randomness used to compute $E_{\mathsf{N}}(\mathsf{otp}_P^{\mathsf{b}})$ and keeps only $\mathsf{otp}_P^{\mathsf{b}}$. The YOSO MPC generating the keys will take $E_{\mathsf{N}}(\mathsf{otp}_P^{\mathsf{b}})$ as input and post $c_P^{\mathsf{b}} = \mathsf{gsk}_P^{\mathsf{b}} \oplus \mathsf{otp}_P^{\mathsf{b}}$ on the TOB. In the simulation we can equivocate by lying about $\mathsf{otp}_P^{\mathsf{b}}$.

The ideal functionality is given in Fig. 6. Note that we leak sp to the adversary. This means that encryption under N is not secure in a context where $\mathcal{F}_{\text{RA+MPC+CF}}$ is *used* as hybrid functionality. This is fine, as we only need that encryption under N is secure when we *implement* $\mathcal{F}_{\text{RA+MPC+CF}}$. Note, in particular that encryption under the global keys gpk_i and role keys rpk_i are still secure. The reason why we leak sp is that when we prove security we need the simulator to know sp to be able to do straight-line simulation.

Note that when a party P is corrupted leakage of the secret key $\mathsf{gsk}_P^{\mathsf{b}}$ happens between batch b being ordered by the first honest party, where $\mathsf{gsk}_P^{\mathsf{b}}$ may generated by the adversary in **Generate Batch**, and batch b being delivered at P, where gsk_P^{b} is deleted. It models that in the implementation, from when c_P^{b} is posted on the TOB a corruption of P will leak $\mathsf{otp}_P^{\mathsf{b}}$ and hence $\mathsf{gsk}_P^{\mathsf{b}}$. Once the TOB delivered at P the party P may choose to delete $\mathsf{gsk}_P^{\mathsf{b}}$. For this to be effective it is important the $\mathcal{F}_{\text{RA+MPC+CF}}$ deleted it too.

We also consider a multi-role version $\mathcal{F}_{\text{RA+MPC+CF}}^{\mathsf{PEAS},\epsilon,\mathbb{F},\gamma,m}$ which assigns m roles to each P per batch. It uses the same $\mathbb{P}^{\mathsf{b}}$ for all batches, but generates separate $\mathsf{gpk}^{\mathsf{b},k}, \pi^{\mathsf{b},k}$, and $\mathsf{rpk}^{\mathsf{b},k}$ for $k = 1, \ldots, m$. This is for uses where M is small and many committees are needed per batch.

The ideal functionality also allows to evaluate a function securely. Since the network is asynchronous we cannot ensure that all parties can give input. We therefore again give a wait predicate W. This is a monotone predicate which tells whether we can evaluate f after having seen inputs form parties in Q, i.e., if $W(Q) = \top$ then $W(Q \cup Q') = \top$. All parties are assumed to agree on W in a given round and the function f to compute. Since each batch of role assignment only generates a bounded number of roles, and the YOSO MPC used to

MPC f consumes a number of roles depending on f, there is some set $\mathbb{F}$ of feasible function. The ideal functionality $\mathcal{F}_{\text{RA+MPC+CF}}$ again leaks the outputs of P until they were delivered. To get a simpler presentation we look only at functions outputting to roles and to all roles. Finally $\mathcal{F}_{\text{RA+MPC+CF}}$ also generates a number of coin-flips which can be revealed later.

Functionality $\mathcal{F}_{\text{SETUP}}$ for setup parametrised by a number of parties M, commitment key generator Gen, PEAS, committee size n, initial number of committees eno

1. Generate $(\mathsf{N}, \mathsf{sp}) \leftarrow \mathsf{PEAS.Gen}$, $\mathsf{ck} \leftarrow \mathsf{Gen}$, Let $S = \log(M)$, $K = (K_1, \ldots, K_{\log N})$ be a random bit-string, $w_S \leftarrow (N+1)u^{N^S} \bmod N^{S+1}$ [a], $c^* \leftarrow E_{N,w_1}(0)$, $c_{K_i} = E_{N,w_1}(K_i)$, $i = 1, \ldots, \log N$, and output $(\mathsf{N}, \mathsf{ck}, w_S, c^*, \{c_{K_i}\})$ to all parties.
2. For $c = 1, \ldots, \mathsf{eno}$ do:
   (a) Sample n unique random parties $P_1^c, \ldots, P_n^c$ from $\mathbb{P}$.
   (b) For $1 \in [n]$ generate $(\mathsf{gpk}_i^c, \mathsf{gsk}_i^c) \leftarrow \mathsf{PEAS.Gen}(\mathsf{N})$, $\mathsf{rpk}_j^c \leftarrow \mathsf{Ran}(\mathsf{gpk}_i^c)$.
   (c) Sample $(\mathsf{pv}^c, \mathsf{sv}_1^c, \ldots, \mathsf{sv}_n^c) \leftarrow \mathsf{VSS}(\mathsf{sp})$ [b].
   (d) Output $(\mathsf{pv}^c, \mathsf{rpk}_1^c, \ldots, \mathsf{rpk}_n^c)$ to all parties.
   (e) For $i \in [n]$ output $(\mathsf{gsk}_i^c, \mathsf{sv}_i^c)$ to P_i^c.

[a] A Paillier encryption of 1 serving as base for other encryptions.
[b] The VSS produces a public part pv and a secret part sv for each party, see Section 6.1 for details.

Fig. 7. Setup for implementing $\mathcal{F}_{\text{RA+MPC+CF}}$

Setup In our implementation we use RA to implement threshold decryption for $\mathsf{Dec}_{\mathsf{sp}}$ and threshold decryption to implement RA. To get off the ground we need that there are some initial RA committees and that the initial committees are given a secret sharing of sp. We assume these secret sharings are given by some setup phase modelled by $\mathcal{F}_{\text{SETUP}}$. The setup also contains an encryption of who is whom in the initial batch such that their public keys can be replaced by new ones once the preprocessed batch of roles have executed.

6 Implementing $\mathcal{F}_{\text{RA+MPC+CF}}$

In this section we give an implementation of $\mathcal{F}_{\text{RA+MPC+CF}}$. In the following we will define several protocols of similar form: all members of a committee send a message of some form to $\mathcal{F}_{\text{TOB}}$ including a zero-knowledge proof that the message was computed correctly. In all such cases, the wait predicate will say that $\mathcal{F}_{\text{TOB}}$ should wait for at least $n-t$ messages from the committee, of the correct form, where the zero-knowledge proofs verify. Here, $t < n/2$ is the bound we assume on the number of corrupted players in the committee. In the protocol descriptions "the message is sent to $\mathcal{F}_{\text{TOB}}$" tacitly implies the above.

We will assume that when $\mathcal{F}_{\text{TOB}}$ delivers a message from some role, the message was indeed sent to $\mathcal{F}_{\text{TOB}}$ by the player having the role. Note that the relevant player needs to sign the message using the secret key corresponding to the role, which has its public key on $\mathcal{F}_{\text{TOB}}$. The only way the assumption could fail is if a corrupt player forges the signature of an honest player.

We use some basic subprotocols for secure computation on ciphertexts of form $E_{N,w_s}(m;r)$. They include protocols for secure multiplication on encrypted values and creating random encrypted bits. As these are mostly standard, we list them in Appendix I. We will use the following subprotocols.

- $\mathsf{Multiply}(E_{N,w_s}(x_1),\ldots,E_{N,w_s}(x_l)) = E_{N,w_s}(x)$ is shorthand for invoking the multiplication protocol a number of times on the ciphertexts $E_{N,w_s}(x_i)$ to obtain $E_{N,w_s}(x)$ where $x = \prod_i x_i \bmod N^s$.
- $\mathsf{RandExp}(E_{N,w_1}(x_1),\ldots,E_{N,w_1}(x_l)) = (E_{N,w_1}(y_1),\ldots,E_{N,w_1}(y_l))$ is shorthand for a protocol that securely chooses a random exponent a of the same bit length as N and computes output ciphertexts $(E_{N,w_1}(y_1),\ldots,E_{N,w_1}(y_l))$ where $y_i = x_i^a \bmod N$. It also produces a set of ciphertexts $\{c_{a_i} = E_{N,w_1}(a_i)\}$, where the a_i's are the bits in the binary representation of a.
- $\mathsf{Exp}(E_{N,w_1}(x), \{c_{a_i} = E_{N,w_1}(a_i)\})$ is short hand for a protocol that outputs $E_{N,w_1}(x^a \bmod N)$, where a is the number with bits $\{a_i\}$ in its binary representation.

6.1 Protocols for Threshold Decryption and Resharing

The secret decryption exponent will be verifiable secret-shared among the members of a committee using *Linear Integer Secret Sharing* (LISS). Using LISS instead of the integer version of Shamir sharing will allow us to keep the size of shares from growing. For details on LISS and the VSS we use, see Appendix F.

The notation $\mathsf{VSS}(d, \boldsymbol{v}_d) = (\mathsf{sh}(d, \boldsymbol{v}_d), \beta_1, \ldots \beta_c)$ means the following: The *sharing vector* $\boldsymbol{v}_d$ contains the secret d as its first entry and the other entries are the randomness used for the sharing. The β_j are commitments to the entries in $\boldsymbol{v}_d$, so β_1 determines the secret. $\mathsf{sh}(d, \boldsymbol{v}_d)$ is the vector of shares resulting from applying the sharing algorithm to $\boldsymbol{v}_d$ which concretely means multiplying $\boldsymbol{v}_d$ by a public matrix M with integer entries. Therefore, given the β_j's and M one can compute commitments $\alpha_i = \mathsf{Com}_{\mathsf{ck}}(s_i; v_i)$ to the i'th share s_i from $\mathsf{sh}(d, \boldsymbol{v}_d)$, using the homomorphic property of the commitments.

A secret will always be shared among the members of a pair of committees, namely it is additively shared among the members of the *additive committee*, and each additive share is shared with threshold t among the members of the *threshold committee*. This is described by a single linear sharing algorithm M.

We number the shares in $\mathsf{sh}(d, \boldsymbol{v}_d)$ such that the first n entries are the additive shares. Further, for $1 \leq i \leq n$ and a qualified set A of the threshold committee, we let $\boldsymbol{r}_A^i$ denote the reconstruction vector that players in A can use to reconstruct the additive share $s_i = \mathsf{sh}(d, \boldsymbol{v}_d)[i]$, while $\boldsymbol{r}_A$ is the reconstruction vector used to reconstruct the secret itself. I_i contains the set of indices of elements in $\mathsf{sh}(d, \boldsymbol{v}_d)$ that are threshold shares of the additive share s_i. Finally, each player in the threshold committee gets several shares, we therefore use $u(i)$ to denote the index of the player holding the i'th share.

We say that a committee pair *holds* $\mathsf{VSS}(d, \boldsymbol{v}_d) = (\mathsf{sh}(d, \boldsymbol{v}_d), \beta_1, \ldots \beta_c)$, *with share bound* b if it is the case that the honest parties in the committee hold the relevant shares from $\mathsf{sh}(d, \boldsymbol{v}_d)$, the corresponding values $\beta_1, \ldots \beta_c$ are on the ledger, and all shares in $\mathsf{sh}(d, \boldsymbol{v}_d)$ are at most 2^b. Note that, by the linearity property of the sharing scheme, if a committee pair

holds both $\mathsf{VSS}(d, \boldsymbol{v}_d) = (\mathsf{sh}(d, \boldsymbol{v}_d), \beta_1, \ldots \beta_c)$ and $\mathsf{VSS}(d, \boldsymbol{v}_d) = (\mathsf{sh}(d', \boldsymbol{v}_{d'}), \beta'_1, \ldots \beta'_c)$, we can define linear operations on these, for a public integer γ, as

$$\mathsf{VSS}(d, \boldsymbol{v}_d) + \gamma \cdot \mathsf{VSS}(d', \boldsymbol{v}_{d'}) = (\mathsf{sh}(d, \boldsymbol{v}_d) + \gamma \cdot \mathsf{sh}(d', \boldsymbol{v}_{d'}), \beta_1(\beta'_1)^\gamma, \ldots \beta_c(\beta'_c)^\gamma)$$

Clearly this new object is also a VSS, we have

$$\mathsf{VSS}(d, \boldsymbol{v}_d) + \gamma \cdot \mathsf{VSS}(d', \boldsymbol{v}_{d'}) = \mathsf{VSS}(d + \gamma d', \boldsymbol{v}_d + \gamma \boldsymbol{v}_{d'}) \ .$$

It follows that if a committee pair holds several VSS objects, it also holds (by local operations on shares) a VSS containing a given linear function applied to the underlying secrets, with a share bound that is easy to compute using the coefficients in the linear function.

Discussion of the Decryption Protocol For the implementation of threshold decryption, we assume a committee pair assigned to handle each batch of ciphertexts to decrypt. We maintain the invariant that when a committee pair is about to decrypt a batch, it holds $\mathsf{VSS}(d_S, \boldsymbol{v}_{d_S})$ with share bound 2^b, where d_S is the decryption exponent. This is ensured for the first pair by $\mathcal{F}_{\text{SETUP}}$, and later by resharing d_S for the next pairs. We split decryption and resharing in two protocols, with one committee pair assigned to do only decryption or resharing.

We prove in Appendix K that the invariant is maintained and that hence the decryption protocol outputs correct plaintexts. The idea for decryption is that each member of the additive committee issues a decryption message by raising the ciphertext to its share. These messages are distributed to all parties using a protocol called Gather.[4] It ensures than when a set of n parties (of which at most t are corrupt) all try to send a message to all parties, all receivers will get messages from at least $n - t$ senders, and there will be a set of at least $n - t$ senders that all receivers have heard from. For details, see Section C.6. Once the Gather protocol is done, we let the threshold committee supply back-up messages that allow reconstruction of all the missing decryption messages. We use this set-up, rather than a single threshold committee, for technical reasons, in order to be able to show adaptive security. For similar reasons, an initial subprotocol is included, RandomizeCiphertext (Fig. 8). When decrypting values that serve as output from the global protocol, we call RandomizeCiphertext on the input ciphertext before the actual decryption. This is a trick that allows us to get the correct decryption result in the simulation. For all other decryption operations, the call to RandomizeCiphertext is omitted, we denote this by DecryptNoRandomize. For input $\bar{c}$, Randomize($\bar{c}$) will denote the ciphertext output from this protocol[5].

Keeping the Share Size Constant In the Reshare protocol we showed a simple version where the share sizes grow with the number of reshares. Here, we sketch a variant of the reshare protocol that allows us to ensure that the shares held by committees are of fixed size and will not grow indefinitely.

[4] For efficiency, the decryption protocol does not use $\mathcal{F}_{\text{TOB}}$ to communicate its output, instead we borrow a more efficient subprotocol from the implementation of $\mathcal{F}_{\text{TOB}}$.

[5] This subprotocol uses the multiplication protocol, which in turn uses decryption. In order for this not to become circular, the decryptions in the multiplication protocol are done without the randomization step.

Protocol RandomizeCiphertext Recall that c^* is the ciphertext from the global set-up and H is the random oracle, here assumed to output a random number from $\mathbb{Z}_{N^{s+1}}$. Also, we assume L is a unique label assigned to this batch of ciphertexts[a]. When a batch to decrypt appears on the ledger, do the following for each ciphertext $\bar{c}$ to decrypt (in parallel):

1. Call the DecryptNoRandomize protocol on input $H(L)$ and let R be the resulting plaintext.
2. Output $c = \bar{c} \cdot \mathsf{Multiply}(c^*, H(\bar{c}, R)) \bmod N^{s+1}$.

[a] no randomness is required here, L could just be a counter, for instance.

Fig. 8. The RandomizeCiphertext protocol.

Protocol Decrypt
When a batch to decrypt appears on the ledger, the committee pair assigned to do the batch will execute the operations below. The committee pair holds $\mathsf{VSS}(d_S, \boldsymbol{v}_{d_S})$ with share bound 2^b.

1. For each ciphertext $\bar{c}$ in the batch, set $c = \mathsf{Randomize}(\bar{c})$.
2. For each member of the additive committee P_i, each $c = E_{N,w_s}(m)$ in the batch and share $s_i = \mathsf{sh}(d_S, \boldsymbol{v}_{d_S})[i]$, P_i sets:

$$d_{c,i} = c^{s_i} \bmod N^{s+1}$$
$$\pi_{c,i} = \mathsf{NIZK}(s_i, v_i : d_{c,i} = c^{s_i} \bmod N^{s+1}, \alpha_i = \mathsf{Com}_{\mathsf{ck}}(s_i; v_i)) \ ,$$

   and send the *decryption message* $(d_{c,i}, \pi_{c,i})$ to all parties.
3. All parties: Once $n - t$ decryption well-formed decryption messages have been received, run Π_{GATHER} (cf. Fig. 17) with the received set as input. The input is justified by consisting of at least $n - t$ well-formed decryption shares.
4. For each member P_u of the threshold committee: Let D be the set of all decryption messages finally received via Π_{GATHER}. Send D to all parties. In addition, for each additive share s_i for which a decryption message was not received, for each share s_j where $j \in I_i, u(j) = u$, and for each ciphertext c in the batch, set:

$$d_{c,i,j} = c^{s_j} \bmod N^{s+1} \ ,$$
$$\pi_{c,i,j} = \mathsf{NIZK}(s_j, v_j : d_{c,i,j} = c^{s_j} \bmod N^{s+1}, \ \alpha_j = \mathsf{Com}_{\mathsf{ck}}(s_j; v_j))$$

   and send the *back-up message* $(d_{c,i,j}, \pi_{c,i,j})$ to all parties.
5. All parties: Once messages from a subset A (of size at least $n-t$ parties) of the threshold committee have been received, then do the following for each ciphertext $c = E_{N,w_s}(m)$ in the batch: For each additive share s_i where a decryption message $d_{c,i}$ was not received, use the reconstruction vector $\boldsymbol{r}_A^i$ to compute

$$d_{c,i} = \prod_{j, P_{u(j)} \in A} d_{c,i,j}^{\boldsymbol{r}_A^i[j]} \bmod N^{s+1},$$

   then compute

$$\prod_{i=1}^{n} d_{c,i} = (N+1)^m \bmod N^{s+1}$$

   and compute m from this result.

Fig. 9. The Decrypt protocol.

Protocol Reshare

1. For each member P_u of the threshold committee, and each share $s_i = \mathsf{sh}(d_S, \boldsymbol{v}_{d_S})[i]$ for which $u(i) = u$, P_u sets $\beta_1^i = \alpha_i$, as defined by $\mathsf{VSS}(d_S, \boldsymbol{v}_{d_S})$ held by the committee. Then compute and sends to $\mathcal{F}_{\text{TOB}}$:

$$\mathsf{NonIntVSSshare}(2^b, s_i, \boldsymbol{v}_{s_i}, \mathsf{gpk}_1, \ldots, \mathsf{gpk}_{2n}) := (\beta_1^i, \ldots, \beta_c^i, c_1^i, \ldots, c_{2n}^i, \pi_0^i, \pi_1^i, \ldots, \pi_{2n}^i),$$

   where $\mathsf{gpk}_1, \ldots, \mathsf{gpk}_{2n}$ are the public keys of the next committee pair assigned to do decryption.
2. Each party P_u in the receiving committee pair: once at least $n - t$ VSS messages from a subset A are delivered from $\mathcal{F}_{\text{TOB}}$, decrypt all ciphertexts c_j^i for which $u = u(j)$. As argued elsewhere, the committee now holds $\mathsf{VSS}(s_i, \boldsymbol{v}_{s_i})$ for i such that $P_{u(i)} \in A$. The committee uses the reconstruction vector $\boldsymbol{r}_A$ to compute

$$\mathsf{VSS}(d_S, \sum_i \boldsymbol{r}_A[i] \cdot \boldsymbol{v}_{s_i}) = \sum_i \boldsymbol{r}_A[i] \cdot \mathsf{VSS}(s_i, \boldsymbol{v}_{s_i}) \ .$$

Fig. 10. The Reshare protocol.

We will amend the VSS protocol such that the dealer must give zero-knowledge proofs for the commitments β_j to the sharing vector, that the number contained in each β_j is in the expected range for entries of the sharing vector. This way, whenever a committee holds $\mathsf{VSS}(d_S, \boldsymbol{v}_{d_S})$, we are guaranteed a publicly known bound on the size of the entries in $\boldsymbol{v}_d$.

The protocol involves 3 committee pairs: the sending pair C_1, an auxiliary pair C_2 and the receiving pair C_3. C_1 holds $\mathsf{VSS}(d_S, \boldsymbol{v}_{d_S})$. The members of C_1 will reshare d_S to C_2 as we already described, so that C_2 will eventually hold $\mathsf{VSS}(d_S, \boldsymbol{v}'_{d_S})$ for some sharing vector $\boldsymbol{v}'_{d_S}$. In addition, each threshold member P_u of C_1 will choose a random integer R_u that is k bit longer than d_S. They then use $\mathsf{NonIntVSSshare}$ to send $\mathsf{VSS}(R_u, \boldsymbol{a}_u)$ to C_2. Finally each member also sends $\mathsf{VSS}(R_u, \boldsymbol{b}_u)$ to C_3. It is easy to ensure that the same R_u is sent in both cases, P_u must use the same commitment β_1 to R_u for both VSS's. Importantly, the sharing vector $\boldsymbol{a}_u$ is chosen large enough that the entries are k bit longer than any entry in $\boldsymbol{v}'_{d_S}$. On the other hand, $\boldsymbol{b}_u$ is chosen of minimal size for sharing R_u, i.e. with entries that are $2k$ bits longer than d_S.

Now, the threshold members C_2 will publicly reconstruct $\mathsf{VSS}(d_S, \boldsymbol{v}'_{d_S}) - \sum_u \mathsf{VSS}(R_u, \boldsymbol{a}_u)$ where the sum over the (at least $n-t$) VSS's that were delivered on the ledger. Concretely, this means posting the corresponding shares on the ledger. This allows all players to reconstruct $\Delta = d_S - \sum_u R_u$. In particular, all members of C_3 can now compute (locally)

$$\mathsf{VSS}(d_S, \boldsymbol{v}_\Delta + \sum_u \boldsymbol{b}_u) = \mathsf{VSS}(\Delta, \boldsymbol{v}_\Delta) + \sum_u \mathsf{VSS}(R_u, \boldsymbol{b}_u) \ ,$$

where $\boldsymbol{v}_\Delta$ is a fixed default sharing vector, say with 0 entries, except for Δ.

This works, first because the shares revealed when reconstructing Δ are determined by the sharing vector $\boldsymbol{v}'_{d_S} - \sum_u \boldsymbol{a}_u$, which is statistically indistinguishable from a vector that is independent of d_S, so essentially no information on d_S is revealed. And second because the size of the final shares in $\mathsf{VSS}(d_S, \boldsymbol{v}_\Delta + \sum_u \boldsymbol{b}_u)$ is fixed, given the secret sharing scheme, n and d_S, and so does not depend on the sizes of shares we started from.

YOSO MPC of the PIR Protocol. For implementation of role assignment, we start by making a useful observation we are going to use below: Say we are given 2 ciphertexts of form $E_{N,w_s}(E_{N,w_{s-1}}(m_1; r_1); u_1)$ and $E_{N,w_s}(E_{N,w_{s-1}}(m_2;_2 r); u_2)$. If we were to apply the multiplication protocol to these two ciphertexts, we would obtain

$$E_{N,w_s}(E_{N,w_{s-1}}(m_1; r_1) \cdot E_{N,w_{s-1}}(m_2; r_2) \bmod N^s; u_3) = E_{N,w_s}(E_{N,w_{s-1}}(m_1 + m_2; r_1 r_2); u_3),$$

thus implicitly adding the two underlying plaintexts.

Protocol RandKey

1. Take the next available set of encrypted bits $c_0 = E_{N,w_1}(t_0), \ldots, c_{\lambda-1} = E_{N,w_\lambda}(t_{\lambda-1})$ from the ledger. Let t be the number represented by the bits $t_0, \ldots, t_{\lambda-1}$, where t_0 is the most significant bit. We will use $c = E_{N,w_*}(t) = (c_0, \ldots, c_{\lambda-1})$ to denote the bitwise encryption of t.
   Let $\mathsf{gpk}_1, \ldots \mathsf{gpk}'_M$ be the current public keys of all parties, with encryption components $h_1, \ldots, h_{M'}$ and signature components $\tilde{h}_1, \ldots, \tilde{h}_{M'}$. The next steps are done using $h_1, \ldots, h_{M'}$ as input. They are repeated in parallel using $\tilde{h}_1, \ldots, \tilde{h}_{M'}$ as input, but the same $c_0, \ldots c_{\lambda-1}$
2. All parties executes the server side of the PIR protocol using $h_1, \ldots, , h_M$ as the database. Note that each h_i is a single number modulo N. The output is a ciphertext $c = E_{N,w_{\lambda-1}}(E_{N,w_{\lambda-2}}(\ldots E_{N,w_1}(h_t)))$.
3. Set $\tilde{c}_{\lambda-1} = c$ and execute the following loop, for $i = \lambda - 1, \lambda - 2, \ldots, 2$ using $2(\lambda - 2)$ consecutive committees:
   (a) Take the next $d_{i-1} = E_{N,w_i}(E_{N,w_{i-1}}(0; r); s)$ available on the ledger, and set $u_i = \mathsf{Multiply}(\tilde{c}_i, d_{i-1})$.
   (b) Send u_i to the Decrypt protocol and let $\tilde{c}_{i-1}$ be the result. We have $\tilde{c}_{i-1} = E_{N,w_{i-1}}(0; r) \cdot p_i \bmod N^i$, where p_i is the plaintext contained in $\tilde{c}_i$, so $\tilde{c}_{i-1}$ is a random ciphertext with the same content as p_i.
4. The final ciphertext produced by the loop is a random ciphertext $\tilde{c}_1$ such that $\tilde{c}_1 = E_{N,w_1}(h_t)$. Let $v = E_{N,w_1}(g)$ be an encryption of g with randomness 1. Run $\mathsf{RandExp}(\tilde{c}_1, v)$ to get ciphertexts $E_{N,w_1}(h_t^a \bmod N)$ and $E_{N,w_1}(g^a \bmod N)$ for random exponent a. Send these two ciphtertexts to the Decrypt protocol to get the random and randomized public key component $(h_t^a \bmod N, g^a \bmod N)$.
   By parallel repetition we will also output $(\tilde{h}_t^{a'} \bmod N, g^{a'} \bmod N)$.
5. Output the key $\mathsf{rpk}_t = ((h_t^a \bmod N, g^a \bmod N), (\tilde{h}_t^{a'} \bmod N, g^{a'} \bmod N))$.

Fig. 11. The protocol for randomized keys

We now design a protocol outputing a random and randomized public key, taken from the global set of public keys $\mathsf{gpk}_1, ..., \mathsf{gpk}_{M'}$, corresponding to a subset of the parties, such that the communication complexity is sublinear in M. We do this by modifying the PIR protocol from Section 3, so it can be executed by a set of committees, where we think of the set of public keys as the database, and a (set of) committees play the role of the client. Recall, however, that each public key consists of two components $h_i, \tilde{h}_i$ for encryption, and signatures, respectively. For technical reasons, we need to randomize the two components individually, so we run two instances of PIR, one for retrieving a random h_i and one to get a random $\tilde{h}_i$, while making sure the random choice is the same in both cases.

More concretely, the high-level idea is that we first use the protocol for encrypted random bits to get what corresponds to the client's first message in the PIR protocol,

$$c_0 = E_{N,w_1}(t_0), \ldots, c_{\lambda-1} = E_{N,w_\lambda}(t_{\lambda-1}) \ ,$$

where the t_i's are random bits and in this application $\lambda = \log M'$. Given that these ciphertexts are on the ledger, any party can consider the $h_1, \ldots, h_{M'}$ to be the database and locally execute the server's part of the PIR. This will result in a ciphertext of form $E_{N,w_{\lambda-1}}(E_{N,w_{\lambda-2}}(\ldots E_{N,w_1}(h_t)))$, containing a randomly chosen public key. Since the server side of the PIR is deterministic, any honest party will arrive at the same ciphertext. However, it would be insecure to decrypt this, even partially. For instance, if we directly remove all but one layer of encryption, to get $E_{N,w_1}(h_t)$, this would reveal information on which key is chosen, as this encryption was computed in the first stage of the loop. Concretely, it is of form $E_{N,w_1}(h_t) = c_0^{h_j} \bar{c}_0^{h_{j+M'/2}}$, for some j, where h_t is either h_j or $h_{j+M'/2}$. An adversary can just try all possibilities for j.

We therefore need to randomize the encryption of h_t before decrypting anything. For this purpose, we assume that we have available on the ledger a sufficient number of random "two level" encryptions of 0, concretely random ciphertexts of form $E_{N,w_{i+1}}(E_{N,w_i}(0))$ for $i = 1 \ldots \lambda - 1$. Such a set of ciphertexts can be used to randomize the encryption of h_t before decryption. Having done this, we can do the same protocol for the $\tilde{h}_i$'s while using the same encryptions of the bits t_j to ensure that the choice of t is the same.

The NewRole protocol The $\mathsf{NewRole}$ protocol is an extension of $\mathsf{RandKey}$ tailored for the below role assignment protocol. We start from the assumption that, for some set of players $P_1, \ldots, P_{M'}$, we are given gpk_i, for $i = 1 \ldots M'$.

All players can now compute $H(\mathsf{b}, i)$ for $i = 1 \ldots M'$, where H is the random oracle, here assumed to output a random number in $\mathbb{Z}_N^*$. Here b is a unique batch number decouple the use of H in different batches. The protocol now considers a database with entries of form $(\mathsf{gpk}_i, H(\mathsf{b}, i))$ and does a multiparty PIR protocol on this database similar to what we described in detail in the $\mathsf{RandKey}$ protocol. In the $\mathsf{RandKey}$ protocol, we did the PIR twice, with the same choice of random index t, once for each component of gpk_i. We now do it 3 times since we have a total of 3 components in each database entry. Having done what corresponds to Step 3 of $\mathsf{RandKey}$ we will have random encryptions $E_{N,w_1}(h_t), E_{N,w_1}(\tilde{h}_t), E_{N,w_1}(H(\mathsf{b}, i))$.

Finally a randomized key rpk_t is produced from $E_{N,w_1}(h_t), E_{N,w_1}(\tilde{h}_t)$ and output, exactly as in $\mathsf{RandKey}$. In addition $\mathsf{tag}_t = H(t)^K \bmod N$ is output. This last output is produced as follows: K is a random number of length $\log_2(N)$, whose binary representation is given in encrypted form as set-up data. We can therefore do Exp on input $E_{N,w_1}(H(\mathsf{b}, i))$ and encrypted bits $E_{N,w_1}(K_j), i = 0, \ldots, \log_2(N) - 1$ to get $E_{N,w_1}(H(\mathsf{b}, i)^K)$ which is decrypted to get the output. Looking ahead, tag_t is a pseudorandom but unique tag assigned to P_t and can be used to detect if we have selected player P_t in a different run of the protocol, without revealing the identity of P_t. We use $\mathsf{NewRole}^{\mathsf{b}}(\mathsf{gpk}_1, \ldots, \mathsf{gpk}_{M'}) = (\mathsf{rpk}_t, \mathsf{tag}_t)$ as shorthand for a call to this protocol.

The NewKey Protocol. This protocol generates a new key pair for a party. It assumes that an encryption of a one-time pad $E_{N,w_1}(\mathsf{otp})$ from that party appears on the ledger. It generates a public key and and an encryption of the secret key under the one-time pad. Details can be found in the appendix (Fig. 34 and surrounding text). We use the notation $\mathsf{NewKey}(E_{N,w_1}(\mathsf{otp})) = (\mathsf{rpk}, E_{\mathsf{otp}}(\mathsf{gsk}))$.

Protocol $\mathsf{OneBatch}^{\mathsf{eno}}$ parametrised by a number M of parties, PEAS, and a number eno which is an upper bound on the number of committees needed to run $\mathsf{OneBatch}^{\mathsf{eno}}$.

Init:
1. Let $c > 0$ be a constant, called the honesty gap.
2. Let $M = (3+c)T$ be the number of parties, where T is the maximal tolerable corruption.
3. Let $n = \lambda$ and let $\phi = \ell n$ be the largest multiple of n s.t.$\phi \leq (c/2)T$.
4. Learn N from $\mathcal{F}_{\text{SETUP}}$ and output N. Define eno committees each with a secret sharing of sp from the output of $\mathcal{F}_{\text{SETUP}}$.
5. Each $P \in \mathbb{P}$ lets $\mathsf{bo}_P = 0$ and $\mathsf{bd}_P = 0$.

Next Batch: On input (NEXTBATCH) to P, where $\mathsf{bo}_P = \mathsf{bd}_P$, update $\mathsf{bo}_P \leftarrow \mathsf{bo}_P + 1$ and:
1. Broadcast $O_P \leftarrow E_{N,w_1}(\mathsf{otp}_P)$ for uniformly random $\mathsf{otp}_P \in \mathbb{Z}_N$, along with a proof of plaintext knowledge. Deletes the randomness used for $E_{N,w_1}(\mathsf{otp}_P)$ and save otp_P.
2. Let the wait predicate of $\mathcal{F}_{\text{TOB}}$ be that there is a set $\mathbb{Q}$ such that $Q \in \mathbb{Q}$ has broadcast O_Q with a correct proof and $|\mathbb{Q}| \geq M - T$. Wait for $\mathcal{F}_{\text{TOB}}$ and let $\mathbb{P}^{\mathsf{bo}_P} \leftarrow \mathbb{Q}$.
3. For all $Q \in \mathbb{P}^{\mathsf{bo}_P}$ in parallel run $(\mathsf{gpk}_Q^{\mathsf{bo}_P}, G_Q = E_{\mathsf{otp}_Q}(\mathsf{gsk}_Q^{\mathsf{bo}_P})) \leftarrow \mathsf{NewKey}(O_Q)$. Once P sees $(\mathsf{gpk}_P^{\mathsf{bo}_P}, G_P)$ on $\mathcal{F}_{\text{TOB}}$ it computes $\mathsf{gsk}_P^{\mathsf{bo}_P} = E_{\mathsf{otp}_P}(G_P)$, outputs $\mathsf{gsk}_P^{\mathsf{bo}_P}$ and deletes otp_P and $\mathsf{gsk}_P^{\mathsf{bo}_P}$.
4. Let $\mathsf{gpk}^{\mathsf{bo}_P} = \left\{\left(Q, \mathsf{gpk}_Q^{\mathsf{bo}_P}\right)\right\}_{Q \in \mathbb{P}^{\mathsf{bo}_P}}$.
5. For $j = 1, \ldots, \ell = \lambda|\mathbb{Q}|$, run $(\mathsf{tmprpk}_j, \mathsf{tag}_j) \leftarrow \mathsf{NewRole}^{\mathsf{bo}_P}(\mathsf{gpk}^{\mathsf{bo}_P})$.
6. If there is not $|\mathbb{P}^{\mathsf{b}}|$ unique values tag_j, then terminate. Otherwise, sort the outputs $(\mathsf{tmprpk}, \mathsf{tag})$ of $\mathsf{NewRole}$ lexicographically on tag and for each unique tag map the first occurrence $(\mathsf{tmprpk}_j, \mathsf{tag})$ unto a unique role $R \in \mathbb{P}^{\mathsf{b}}$ and let $\mathsf{rpk}_R^{\mathsf{bo}_P} = \mathsf{tmprpk}_j$.
7. Let $\mathsf{rpk}^{\mathsf{bo}_P}$ be $\{(R, \mathsf{rpk}_R^{\mathsf{bo}_P})\}_{R \in \mathbb{P}^{\mathsf{bo}_P}}$. Let $R^{\mathsf{bo}_P} = (\mathsf{gpk}^{\mathsf{bo}_P}, \mathsf{rpk}^{\mathsf{bo}_P})$ and output R^{bo_P}.
8. Let $\mathsf{bd}_P \leftarrow \mathsf{bd}_P + 1$.

Renew Setup: At the same time as the above **Next Batch**, do a parallel of **Next Batch** to generate $M - T$ additional roles. Take the $(c/2)T$ first of these roles and use them to form ϕ committees of size n. If $\phi > \mathsf{eno}$ then only run **Renew Setup** if there are not eno committees left from a previous run. If $\phi < \mathsf{eno}$ do the above $\lceil \mathsf{eno}/\phi \rceil$ in parallel to get a total of eno committees. In both cases, for each of the eno committees run a parallel instance of $\mathsf{Reshare}$ to secret share sp until the committee.

Fig. 12. The protocol for one batch of Role Assignment

Iterative Role Assignment We finally present the role assignment protocol in implements the part of $\mathcal{F}_{\text{RA+MPC+CF}}$ having to do only with role assignment, i.e., no MPC and no Coin-Flip. In Fig. 12 we give the code for a single batch called $\mathsf{OneBatch}^{\mathsf{eno}}$. For each batch each party P broadcasts a new encrypted otp_P and we use NewKey to generate a fresh public key gpk_P along with $E_{\mathsf{otp}_P}(\mathsf{gsk}_P)$, where $E_{\mathsf{otp}_P}(\mathsf{gsk}_P)$ is a one-time pad encryption of gsk_P. Once P sent $E_{N,w_1}(\mathsf{otp}_P)$ it deletes the randomness used for encryption. Once P sees $E_{\mathsf{otp}_P}(\mathsf{gsk}_P)$ it decrypts using otp_P. This way we never have to explain the randomness used for $E_{N,w_1}(\mathsf{otp}_P)$ while $E_{\mathsf{otp}_P}(\mathsf{gsk}_P)$ is in transit. We wait only for $M-T$ encryptions to not deadlock. For the set $\mathbb{Q}$ of parties that got to input otp_P we then generate ℓ random role keys rpk. Each is for a random unknown $P \in \mathbb{Q}$. We also output a tag tag which cannot be linked to P but which is unique for P. We use this to throw away duplicates, such that each P gets one role. If some P does not get a role, i.e., there is less than $M-T$ unique tags, then we abort. This happens with negligible probability $(1-(M-t)^{-1})^{\lambda(M-T)} \to e^{-\lambda}$, so we can ignore it. This process is run enough times in parallel to get eno random committees for the next run of the protocol. When we run a protocol like NewKey some m times in parallel we use the same committees in each run, so it consumes a number of committees constant in m. Ergo, $\mathsf{OneBatch}^{\mathsf{eno}}$ consumes

a number of committees constant in eno. We can therefore set eno large enough to generate enough committees for the next batch without circularity. The reason why we only use $(c/2)M$ roles to form committees is that once we used and executed $(c/2)M - 1$ roles and revealed which parties were behind them, there is still $(M - T) - (c/2)T \geq (2 + (c/2))T$ parties left unrevealed. So, even if the adversary concentrates its corruptions on this set it can corrupt the party behind the last unrevealed role of the last committee with probability at most $1/(2+(c/2))$. Therefore, by a Chernoff bound, the probability that $\geq (1/2)\lambda$ parties out of the $n = \lambda$ parties on a committee are corrupted before they executed their role is $\mathrm{negl}(\lambda)$. Hence each committee has honest majority of executed role except with negligible probability, as required for all our sub-protocols. We let $\mathsf{MoreBatch}^{\mathsf{eno}}$ be the protocol starting with $\mathcal{F}_{\mathrm{SETUP}}^{\mathsf{eno}}$ and doing repeated application of $\mathsf{OneBatch}^{\mathsf{eno}}$.

Putting the Pieces Together Once we implemented the role assignment part of $\mathcal{F}_{\mathrm{RA+MPC+CF}}$ it is trivial to implement the MPC part of $\mathcal{F}_{\mathrm{RA+MPC+CF}}$. Parties contribute their inputs by sending $E_{N,w_1}(x_i)$ and sends along an additonal encrypted otp for receiving the output. Both encryptions are augmented by proofs of plaintext knowledge. Then we use the sub-protocols for bits, adding, and multiplying to evaluate f on the contributed inputs and open encryptions of outputs under the otp. We elaborate on this in Appendix J. Let $\mathcal{F}_{\mathrm{RA+MPC}}$ be $\mathcal{F}_{\mathrm{RA+MPC+CF}}$ with the Flip Coin and Deliver Coin commands removed. What we have specified so far implements $\mathcal{F}_{\mathrm{RA+MPC}}$, the proof of this fact is found in Appendix Appendix K.

We can then implement $\mathcal{F}_{\mathrm{RA+MPC+CF}}$ in the $\mathcal{F}_{\mathrm{RA+MPC}}$-hybrid model. This is done by securely computing a function f which divides the parties into committees and robustly secret shares a random value unto the committee. To flip the coin the committee members all send their shares and check values to all other parties. Details and proof are found in Appendix L.

In our implementation of $\mathcal{F}_{\mathrm{RA+MPC+CF}}$ we assume we have access to an instance of $\mathcal{F}_{\mathrm{TOB}}$. In Section 7 we show how to implement $\mathcal{F}_{\mathrm{TOB}}$ from an ideal functionality $\mathcal{F}_{\mathrm{CF}}$ for coin-flip, which in turn will just be the coin-flip part of $\mathcal{F}_{\mathrm{RA+MPC+CF}}$. This seems cyclical, but importantly, since we open coin-flips by reconstructing robust secret sharings, the implementation of openings of coin-flips on $\mathcal{F}_{\mathrm{RA+MPC+CF}}$ does not use $\mathcal{F}_{\mathrm{TOB}}$. We can therefore let $\mathcal{F}_{\mathrm{RA+MPC+CF}}$ implement a surplus of coin-flips in batch b. During the implementation of batch $\mathsf{b}+1$ these can be used to implemented the instance of $\mathcal{F}_{\mathrm{TOB}}$ used in $\mathsf{b}+1$. The implementation of $\mathcal{F}_{\mathrm{RA+MPC+CF}}$ works equally well if each batch uses a separate $\mathcal{F}_{\mathrm{TOB}}$. This gives a protocol which we call $\mathsf{RAMPCCF}^{\mathsf{PEAS}}$.

Theorem 1. *When for a constant c at most $T < M/(3+c)$ parties are adaptive corrupted and we set $n = \lambda$ then for a large enough constant eno we have that if PEAS is EUF-CMA, SO-IND-CPA, CSO-IND-CPA and CSO-ANON, then $RAMPCCF^{\mathsf{PEAS}}$ securely implements $\mathcal{F}_{RA+MPC+CF}^{\mathsf{PEAS},1/3,\mathbb{F},\gamma}$ in the $(\mathcal{F}_{SETUP}, \mathcal{F}_{TOB}, \mathcal{F}_{ATOMIC\text{-}SEND})$-model with a random oracle. Here $\mathbb{F}$ can be any class of functions with a bounded multiplication complexity over $\mathbb{Z}_N$ and γ can be any polynomial. If $M \geq 2\mathsf{eno}$ and we use the $\mathcal{F}_{TOB}$ from Section 7 then the amortized communication in bits to generate one committee of size $n = \lambda$ is $M\,\mathrm{poly}(\lambda, \log M)$. The amortized complexity of handling one multiplication gate in MPC is also $M\,\mathrm{poly}(\lambda, \log M)$.*

As for the communication note that in each basic run we generate $(c/2)T = \Theta(M)$ roles. To do this we use a number of committees which is $\text{poly}(\lambda, \log M)$ as we have a constant number of runs of parallel protocols each with $\text{poly}(\lambda, \log M)$ rounds. Each committee consists of λ parties, so we consume $\text{poly}(\lambda, \log M)$ roles. To generate $\Theta(M)$ roles a total of $O(M)\,\text{poly}(\lambda, \log M)$ group elements are posted on $\mathcal{F}_{\text{TOB}}$. Therefore the amortized number of group elements posted on $\mathcal{F}_{\text{TOB}}$ per generated committee is $O(\lambda)\,\text{poly}(\lambda, \log M) = \text{poly}(\lambda, \log M)$, and the same holds for generating $\mathsf{eno} \in \text{poly}(\lambda, \log M)$ committees. Since our implementation of $\mathcal{F}_{\text{TOB}}$ uses communication $O(ML) + \text{poly}(\lambda, \log M)$ to broadcast L bits to all parties, the amortized number of bits sent and received by each party to generate eno committees is $\text{poly}(\lambda, \log M)$. Handling one multiplication gate uses a constant number of committees. The reason why we generate $\ell = \lambda|\mathbb{Q}|$ roles in Item 4 in **Next Batch** is that we need each party to be sampled at least once. The overhead of λ can be reduced to $O(\log \lambda)$ for roles produced by $\mathcal{F}_{\text{RA+MPC}}$ and $O(1)$ for internal roles. We discuss this in Appendix A.

7 Consensus

We present a protocol Π_{TOB} securely implementing $\mathcal{F}_{\text{TOB}}$. For implementing $\mathcal{F}_{\text{TOB}}$ we need an ideal functionality for asynchronous coin-flip $\mathcal{F}_{\text{CF}}$. Like for the coin related sub-interface of $\mathcal{F}_{\text{RA+MPC+CF}}$, when the first honest party asks for the next coin it is flipped and shown to the adversary, and after the last honest party asks for the next coin, the adversary must eventually deliver the coin to all honest parties. For later convenience we assume $\mathcal{F}_{\text{CF}}$ has a command (coin-index) telling a party P_i how many coins it received from $\mathcal{F}_{\text{CF}}$ so far.

7.1 Implementing $\mathcal{F}_{\text{TOB}}$ from $\mathcal{F}_{\text{CF}}$

We give a high-level description of the protocol Π_{TOB} instantiating $\mathcal{F}_{\text{TOB}}$. It sequentially runs instances of an Agreement on a Core Set (ACS) protocol Π_{ACS} where the inputs of each party satisfies the wait predicate. The ACS protocol is heavily inspired by the one in [KN23], but with several changes to make it YOSO. The protocol Π_{TOB} uses small committees, but does not use role assignment to sample these committees. This is because role assignment establishes private channels to future roles which would be overkill for Π_{TOB} and heavily dominate its communication and computational complexity. We therefore use simple self-nomination from VRFs as in [GHM$^+$17], which is concretely much more efficient. We elaborate in Appendix C.1. The decentralized nature of sampling the roles for Π_{TOB} via self-nomination means that the committees will be of variable size, and that there is no straightforward way to talk about the index of a role on the committee. Therefore, the ACS protocol in [KN23] cannot be instantiated with the self-nominated committees without some structural changes, because it uses the fact that each party corresponds to an integer in $[M]$ both to describe parties' causal past via M-bit vectors and to elect each leader using a $\log M$-bit coin.

7.2 Agreement on Core Set

In Appendix C we present a modified version of the protocol for ACS from [KN23]. Beyond some syntactical changes, we make the following three changes:

1. For each instance of an activation rule the set of parties who "speak" is a committee specifically elected for that role. This is implemented using sortition on a *unique* identifier associated with that instance of the activation rule as described in Appendix C.1. We do not explicitly define these identifiers, but a natural choice is to concatenate the protocol name, session identifier, and a unique activation rule name.
2. The underlying Causal Cast (CC) primitive is instantiated using a novel RB protocol in Appendix C.3. This allows the CC protocol to give output to all parties, as we by design do not know who is in the next committee.
3. Leader election is separated from CC and implemented directly in the context of graded block selection (Appendix C.7) using an extra round of communication and the coin-flip functionality $\mathcal{F}_{\text{CF}}$ in Fig. 29. The resulting protocol has the property that if a party selects a block with grade 2, then *all possible justified outputs* (cf. Appendix C.2) in the following round have grade 2.
4. By using the justified grade of the block selected in the preceding round, as justifier that we did not yet terminate, we make sure that all parties terminate in adjacent rounds, and that the round number becomes a justified output of the protocol.

The first three changes make the ACS protocol YOSO and compatible with self-nominated committees. The final change allows a substantial optimization in the amount of setup that needs to be recomputed when instantiating $\mathcal{F}_{\text{CF}}$ with a YOSO protocol that requires setup for each coin flip.

Many natural instantiations of the coin flip requires a setup to be computed for each round of the protocol. The number of rounds can be bounded by $O(\lambda)$ but only an expected constant number of setups are actually used. So there is a multiplicative factor $O(\lambda)$ overhead on the communication and computation required to compute setups. We cannot a priori repurpose the unused setups for later rounds, because a party cannot determine from its local view, whether another honest party requested a coin and thus leaked it to the adversary. Exposing the justified output round will allow us to reach agreement on an upper bound on the number of setups that could have been used in each iteration by supplying them as input to the next round. This will in turn allow reducing the number of coin flipping setups being consumed by each decision of Π_{TOB} to expected constant.

7.3 Total-Order Broadcast

We present a protocol Π_{TOB} implementing $\mathcal{F}_{\text{TOB}}$ with ledgers, blocks and wait predicates as defined in Section 5. A straightforward implementation would be to have parties who want to broadcast a message on the TOB send the message to all parties, have an elected committee collect these messages and then, when their local blocks satisfy the wait predicate, propose them in Π_{ACS}. But as each message could be included by multiple proposers this would result

Protocol Π_{TOB} described from the view of party P. We use the definitions of ledger, blocks and wait predicates from Section 5.

Init: Each party P initialises the empty ledger $L_{\mathsf{P}} = \epsilon$, a broadcast index keeping track of how many messages were broadcast by P, $c_{\mathsf{P}} = 0$, a batch index keeping track of how many wait predicates are set for P, $b_{\mathsf{P}} = 0$, and a set of dispersed messages pending inclusion in L, $\mathsf{Pending}_{\mathsf{P}} = \emptyset$. It also initialises two instances of the coin functionality $\mathcal{F}_{\mathrm{CF}}^{0}$ and $\mathcal{F}_{\mathrm{CF}}^{1}$ and corresponding coin counters $\mathsf{coin}_0 = 0$ and $\mathsf{coin}_1 = 0$.

Broadcast message: On input $(\textsc{Broadcast}, \mathsf{mid}, m)$ party $P(\mathsf{mid})$ starts an instance of Π_{RB} on the message m with session identifier $(\textsc{Broadcast}, \mathsf{mid}, c_{\mathsf{P}})$ with the input justifier that Π_{RB} has given output for all sessions $(\textsc{Broadcast}, \mathsf{mid}, c)$ with $c \in [1; c_{\mathsf{P}})$. Finally, it lets $c_{\mathsf{P}} = c_{\mathsf{P}} + 1$.

Schedule message: On output m from an instance of Π_{RB} with session identifier $(\textsc{Broadcast}, \mathsf{mid}, c)$ add $(P(\mathsf{mid}), c, m)$ to $\mathsf{Pending}_{\mathsf{P}}$.

Set Wait Predicate: On input $(\textsc{wait}, W)$ let $b_P \leftarrow b_P + 1$ and let $W^{b_P} = W$.

Deliver: When P has output (C, r) from Π_{ACS} with session identifier $|L| + 1$ and for each pair $(\mathsf{P}', c) \in C$ there is a an entry in $\mathsf{Pending}_{\mathsf{P}}$ of the form (P', c, m) it does the following: Adds each of the messages m (ignoring duplicates) in order to a block which is added to L_{P} and removes the corresponding entries from $\mathsf{Pending}_{\mathsf{P}}$. Lets $\mathsf{LocalOutputRound}_{|L|+1} = r$, and lets r' be the minimal justified output round included in C. It then adds $r' + 1$ to $\mathsf{coin}_{|L|+1 \mod 2}$ and inputs $(\textsc{next-coin})$ to $\mathcal{F}_{\mathrm{CF}}^{|L|+1 \mod 2}$ until querying it for input $(\textsc{coin-index})$ returns $\mathsf{coin}_{|L|+1 \mod 2}$.[a]

Propose Block: When $|\mathsf{Pending}_{\mathsf{P}}| \geq \max(W_{\#}^{|L_{\mathsf{P}}|+1}, \alpha)$ and $W^{|L_{\mathsf{P}}|+1}(L_{\mathsf{P}}, \mathsf{Pending}_{\mathsf{P}}) = \top$, P starts running Π_{ACS} with session id $|L_{\mathsf{P}}|+1$. The input block is defined as follows: We say an element $(\mathsf{P}', c, m) \in \mathsf{Pending}_{\mathsf{P}}$ is referenced by (P', c). Party P computes a block B consisting of references to at most $W_{\#}^{|L_{\mathsf{P}}|+1}$ messages in $\mathsf{Pending}_{\mathsf{P}}$ where $W^{|L_{\mathsf{P}}|+1}(L_{\mathsf{P}}, B)$[b] and remove those messages from $\mathsf{Pending}_{\mathsf{P}}$. Then add references to the $\max(W_{\#}^{|L_{\mathsf{P}}|+1}, \alpha, \mathsf{Pending}_{\mathsf{P}})$ oldest messages from $\mathsf{Pending}_{\mathsf{P}}$ to a block B' and remove those messages from $\mathsf{Pending}_{\mathsf{P}}$. Finally let B'' be a block including only the locally observed output round from the previous batch, $\mathsf{LocalOutputRound}_{|L_{\mathsf{P}}|}$ and let $B''' = B \| B' \| B''$ be the input to Π_{ACS}. The block is justified by consisting of a justified output round number from the previous iteration of Π_{ACS} and references to messages that are sent through RB in the step above, these messages satisfying the wait predicate and the number of references being in the interval $[\max(W_{\#}^{|L_{\mathsf{P}}|+1}, \alpha); \max(W_{\#}^{|L_{\mathsf{P}}|+1}, \alpha)]$.

[a] In plain English, it uses the agreement on how many coins were consumed in past iterations to skip past any coins that it has not used but which potentially could have been used by other parties, so that all parties are synchronised when they start flipping coins in the next instance of Π_{ACS}.

[b] This can be computed efficiently as described in Section 5.

Fig. 13. Total-Order Broadcast

in a worst-case multiplicative communication overhead of $O(n)$. Instead we will have each party who wants to broadcast a message on the TOB send the message through reliable broadcast with a $O(\log M)$ bit message identifier, mid, and then have the proposers include the message in their block by referring to the message using mid. Each block proposer will also add the round number it got as part of the output of Π_{ACS} in the previous round to its block. After agreement on a subset of the blocks is reached, we can take the minimum output round, r, included in the set of blocks and via Adjacent Output Round property conclude that no honest party participated in round $r + 2$ or later. Thus the corresponding coins remain unpredictable to the adversary and their setups can be repurposed.

We require that all blocks have the same size up to a constant factor. Otherwise, the communication in each round could be dominated by a large block which does not make it into the core. To get around this issue, we require that all blocks include references to

between $(\max(\alpha, W_{\#}))$ and $2(\max(\alpha, W_{\#}))$ messages, which means that the size of blocks that get included in the core in each epoch is not asymptotically dominated by the remaining blocks. For the concrete complexity analysis in Theorem 5 we assume that α is at least λ.

Theorem 2. *When for a constant c at most $T < M/(3+c)$ parties are adaptive corrupted and we sample committees as in Lemma 2, then we have that Π_{TOB} implements $\mathcal{F}_{TOB}$ in the $\mathcal{F}_{CF}$-model.*

Proof. Follows from Theorem 6, Theorem 5, and Lemma 2.

References

Bai16. Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. SWIRLDS TECH REPORT SWIRLDS-TR-2016-01, 2016. `https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf`.

BKLZL20. Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. pages 353–380, 2020.

BLZLN22. Amey Bhangale, Chen-Da Liu-Zhang, Julian Loss, and Kartik Nayak. Efficient adaptively-secure byzantine agreement for long messages. pages 504–525, 2022.

Bra87. Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, nov 1987.

CCL15. Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. pages 3–22, 2015.

CDN01. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. pages 280–299, 2001.

DF02. Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology—ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology and Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings 8*, pages 125–142. Springer, 2002.

DFK+06. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006.

DJN10. Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9:371–385, 2010.

DN03. Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. pages 247–264, 2003.

DT06. Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In *International Workshop on Public Key Cryptography*, pages 75–90. Springer, 2006.

DXR21. Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. pages 2705–2721, 2021.

FHKP16. Georg Fuchsbauer, Felix Heuer, Eike Kiltz, and Krzysztof Pietrzak. Standard security does imply security against selective opening for Markov distributions. pages 282–305, 2016.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Annual International Cryptology Conference*, pages 152–168. Springer, 2005.

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero-knowledge protocols to prove modular polynomial relations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 82(1):81–92, 1999.

GHK+21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. pages 64–93, 2021.

GHM+17. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. `https://eprint.iacr.org/2017/454`.

GHM+21. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR and applications. pages 32–61, 2021.

KKNS21. Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021.

KN23. Simon Holmgaard Kamp and Jesper Buus Nielsen. Byzantine agreement decomposed: Honest majority asynchronous total-order broadcast from reliable broadcast. *IACR Cryptol. ePrint Arch.*, page 1738, 2023.

LN23. Julian Loss and Jesper Buus Nielsen. Early stopping for any number of corruptions. *IACR Cryptol. ePrint Arch.*, page 1813, 2023.

Nie02. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. pages 111–126, 2002.

ORV14. Rafail Ostrovsky, Vanishree Rao, and Ivan Visconti. On selective-opening attacks against encryption schemes. pages 578–597, 2014.

A Concrete Optimisations

The reason why we generate $\ell = \lambda|\mathbb{Q}|$ keys in Item 4 in **Next Batch** in Fig. 12 is that each $P \in \mathbb{Q}$ needs to be sampled at least once in the PIR so that it gets a role, as $\mathcal{F}_{\text{RA+MPC}}$ by definition outputs a role for each party in $\mathbb{Q}$. The above method is very wasteful. We generate $\ell = \lambda|\mathbb{Q}|$ keys to output only $|\mathbb{Q}|$ roles, and overhead of λ. Note, however, that if one generates $|\mathbb{Q}|$ keys instead of $\lambda|\mathbb{Q}|$ keys, then a given P gets a role with constant probability. Furthermore, we can use the tags to compute how many parties got a role key. Repeating this in sequence until there are $|\mathbb{Q}|$ unique parties which got a role key will take expected $O(\log|\mathbb{Q}|)$ rounds. And we have that $O(\log|\mathbb{Q}|) = O(\log M) = O(\log\lambda)$, as M is polynomial in λ. This replaces an overhead λ by $\log\lambda$. Furthermore, for the roles used to form committees in **Renew Setup** we only use $\phi \leq (c/2)T$ roles out of the $M - T = (2 + (c/2))T$ roles generated. There is no reason to generate role keys for the parties in $\mathbb{Q}$ which are not used to form committees. If we generate $|\mathbb{Q}| = M - T$ random role keys, then by a Chernoff bound, we will have $(c/2))T$ unique roles except with negligible probability, saving an additional $\log M$ factor for internal committees.

B Secure Committee Formation

In the following we let $\mathcal{F}_{\text{RA+MPC}}$ denote a version of $\mathcal{F}_{\text{RA+MPC+CF}}$ without the con-flipping interface. In $\mathcal{F}_{\text{RA+MPC}}$ all parties get to play exactly one role. Often what we want is to get a sequence of random committees

$$((\mathsf{rpk}_{1,1},\ldots,\mathsf{rpk}_{1,n}),\ldots,(\mathsf{rpk}_{\ell,1},\ldots,\mathsf{rpk}_{\ell,n}))$$

for some committee size n. This can be done by using only the first $\psi = n\ell$ roles from $\mathsf{rpk}^{\mathsf{b}}$. For security we typically need that the adversary can corrupt at most $t < n/2$ parties on each committee. This puts some limits on ℓ, as discussed now.

To see the issue, consider the following. When the random committees start executing, some committes may execute first. As the first committee $(\mathsf{rpk}_{1,1},\ldots,\mathsf{rpk}_{1,n})$ executes their roles the adversary will learn which n parties had the roles, as these parties will send the corresponding messages. Similarly for committees $2,\ldots,\ell-1$. This means that when only $(\mathsf{rpk}_{\ell,1},\ldots,\mathsf{rpk}_{\ell,n})$ is left, the adversary knows $(\ell-1)n$ parties $\mathbb{X}$ which already executed their roles. It also knows that the last committee is drawn uniformly at random from $\mathbb{P}^{\mathsf{b}} \setminus \mathbb{X}$. It can therefore concentrate its corruptions on the set $\mathbb{P}^{\mathsf{b}} \setminus \mathbb{X}$. It turns out that if we have $M = (3+c)T$ parties, for a positive gap c, and the adversary can corrupt at most T parties, then we can safely set $\phi \leq (c/2)M$ and $n = \lambda$.

The reader might wonder why we did not use sampling with replacement for the committees to avoid the above complication. The reason is that if a party gets more than one role, it would also need more global public keys to be able to execute them separately and have to send more than one otp for receiving the secret keys. To hide who has multiple roles (they are good targets for corruption) we would then need to generate the worst case number of global keys for all parties and have all parties send the worst case number of one-time pads.

Parameters 1. Let $c > 0$ be a constant, called the honesty gap.
2. Let $M = (3+c)T$ be the number of parties, where T is the corruption budget of A.
3. Let $n = \lambda$ and let $\phi = \ell n$ be the largest multiple of n such that $\phi \leq (c/2)T$.

Run as RA Create an instance of $\mathcal{F}_{\text{RA+MPC}} = \mathcal{F}_{\text{RA+MPC}}^{\mathsf{PEAS},\epsilon=\frac{1}{3+c},1}$. We describe the game for $m = 1$ but it can be played for any adversarially chosen m, i.e., $\mathcal{F}_{\text{RA+MPC}} = \mathcal{F}_{\text{RA+MPC}}^{\mathsf{PEAS},\epsilon=\frac{1}{3+c},m}$. Let A be a PPT adversary interacting with $\mathcal{F}_{\text{RA+MPC}}$ as the environment, with the following exceptions.
- When A inputs $(\text{DELIVERBATCH}, P)$ to $\mathcal{F}_{\text{RA+MPC}}$ for an honest P, then we do not give $(\mathsf{gsk}_P^{\mathsf{bo}_P}, \cdot)$ to A. Instead we save $(P, \mathsf{bo}_P, \mathsf{gsk}_P^{\mathsf{bo}_P})$ for later "execution".
- A may corrupt at most T parties.

Execute Role If at any point A outputs $(\text{EXEC}, b, R \in \mathbb{P}^{\mathsf{b}})$, where for $P = \pi_b^{-1}(R)$ the value (P, b, gsk_P^b) is saved, then give (P, b, gsk_P^b) to the adversary.

Role PE Corruption If at any point A corrupts P then for each $1 \leq b \leq \mathcal{F}_{\text{RA+MPC}}.\mathsf{b}$ let $R_P^b = \pi_b(P)$. If the adversary has not previously given the command (EXEC, b, R_P^b) then we say that rpk_P^b was PE corrupted.

End of Game When A ends the game, we determine a winning bit $w \in \{0,1\}$ as follows. For each $0 \leq b \leq \mathcal{F}_{\text{RA+MPC}}.\mathsf{b}$ let $(\mathsf{rpk}_{1,1}^b, \ldots, \mathsf{rpk}_{1,n}^b, \ldots, \mathsf{rpk}_{\ell,1}^b, \ldots, \mathsf{rpk}_{\ell,n}^b) = (\mathsf{rpk}_1^b, \ldots, \mathsf{rpk}_\phi^b)$. If there exist $0 \leq b \leq \mathcal{F}_{\text{RA+MPC}}.\mathsf{b}$ and $1 \leq j \leq \ell$ such that $\geq n/2$ roles in $(\mathsf{rpk}_{j,1}, \ldots, \mathsf{rpk}_{j,n})$ are PE corrupted, then let $w = 1$. Else, let $w = 0$.

Fig. 14. RA-ANON Game $\mathsf{RAANON}_{\mathsf{PEAS},A}$

This would lead to a significantly larger loss of efficiency than only using cT of the generated roles.

Note that we cannot guarantee that committees have honest majority forever. Once a role R_P executed the adversary knows who P was, so it can in the end corrupt all parties on a committee. However, in all our uses of committees we have that for a corruption of a party to be useful to the adversary, the party must not have executed its role yet. The reason is that when P executes its role R_P then it correctly computes the message m to send, deletes all randomness used to compute it, and then sends m. So, corrupting P after m was sent gives no information extra to m. We call a corruption of P before it executed R_P a pre-execution (PE) corruption of R_P. We say that committee j has PE honest majority if $< n/2$ roles in the committee were PE corrupted. We now formalise a committee corruption game, where the adversary A tries to create a committee which does not have PE honest majority, see Fig. 14.

Definition 4. *We say that* PEAS *is RA-ANON secure if for all PPT* A *it holds that* $\Pr[\mathsf{RAANON}_{\mathsf{PEAS},A}] = 1] = \text{negl}(\lambda)$.

Theorem 3. *Assume that* PEAS *is SO-ANON secure (Definition 3). Then* PEAS *is RA-ANON secure (Definition 4).*

Proof. Note that $|\mathbb{P}^b| \geq M - T$ in all batches with this setting of M, c, and ϵ as the adversary can exclude ϵM parties. For each batch $b \leq \mathcal{F}_{\text{RA+MPC}}.\mathsf{b}$ let $\mathbb{X}^b$ be the set of at most ℓn roles R_P^b executed in that batch.

We can use a simple reduction to SO-ANON security to prove that whenever the adversary corrupts an honest party P which in batch b did not have its role R_P^b executed, i.e., $R_P^b \in \mathbb{P}^b \setminus \mathbb{X}^b$, then for all fixed, unexecuted roles $R \in \mathbb{P}^b \setminus \mathbb{X}^b$ it holds that the probability

that $R_P^b = R$ is negligibly close to $\frac{1}{|\mathbb{P}^b \setminus \mathbb{X}^b|}$. Namely, if this was not the case we could do the following. For all batches b number the corruptions of parties which did not have their roles in batch b executed as $c = 1, 2, \ldots$. When such a corruption happens number all the unexecuted roles as $u = 1, 2, \ldots$, say lexicographically. Then there is a first b, first c and first u such that for batch b when the c'th corruption of a party which did not have their roles in batch b executed happens, then for unexecuted role number u, call it R, it happens that the probability that $R_P^b = R$ is not negligibly close to $\frac{1}{|\mathbb{P}^b \setminus \mathbb{X}^b|}$. We can use (b, c, u) to do a reduction to SO-ANON Security. We let the adversary B of the reduction get a challenge batch from $\mathsf{SOANON}_{\mathsf{PEAS},B}$ and use it as batch b in $\mathsf{RAANON}_{\mathsf{PEAS},A}$. It maps all key leakages from $\mathsf{RAANON}_{\mathsf{PEAS},A}$ for batch b to $\mathsf{SOANON}_{\mathsf{PEAS},B}$. When A does its c'th corruption of a party P_i which did not have its role in batch b executed happens, then B guesses (i, j), where j is the index of the u'th unexecuted role in batch b.

We now use this to argue that PEAS is RA-ANON secure. We need to argue that all committees have $< n/2$ roles in the committee which were PE corrupted. Let b be any batch and j any committees in that batch. Consider any corruption of any P made by the adversary. When it does the corruption it executed at most $\ell n = \phi$ roles from batch b. Let $\mathbb{X}$ be the set of at most ℓn roles executed. Let R be a role from $(\mathsf{rpk}_{j,1}^b, \ldots, \mathsf{rpk}_{j,n}^b)$ and let P be the party having the role. We seek the probability that the adversary corrupts P. Let $\mathbb{Y}^b$ be the parties which have their role executed in batch b already, i.e., $\mathbb{Y}^b = \pi^{-1}(\mathbb{X}^b)$. If the adversary corrupts Q from $\mathbb{Y}^b$ the probability that it hits P is 0 as Q already had an executed role in batch b, so since R is not executed in batch b and Q had at most one role in batch b it cannot also have role R. If the adversary corrupts a party not from $\mathbb{P}^b$, then the party will not have had any role in batch b, so cannot be P, as P has role R. Finally, if the adversary corrupts P from $\mathbb{P}^b \setminus \mathbb{Y}^b$ it follows that the probability that it hits P is $\frac{1}{|\mathbb{P}^b \setminus \mathbb{X}|} \leq \frac{n}{M - T - c/2T} = \frac{1}{((2+(c/2))T}$. For T corruptions the probability that P is corrupted is thus negligibly close to $\frac{1}{2+c/2}$. The expected number of corruptions in committee ℓ is therefore negligibly close to $\frac{n}{2+c/2}$. By a simple Chernoff bound it follows that there are $\geq \frac{n}{2}$ corruptions with probability $2^{-\Theta(\lambda)}$. $\square$

In the real protocol we need RA-ANON security even if functions of the secret key sp is secret shared to the committees. However, this follows from a combination of CSO-IND-CPA security and RA-ANON security, as in RA-ANON the adversary has sp and therefore can simulate these secret sharings itself.

Definition 5 (CSO-RA-ANON). *We say that* $(\mathit{PEAS}, \mathsf{PPBox})$ *is CSO-RA-ANON secure if it holds for all PPT A that*

$$\Pr[\mathsf{CSORAANON}^0_{\mathit{PEAS},\mathsf{PPBox},A} : w = 1] = \text{negl}$$

$$\Pr[\mathsf{CSORAANON}^1_{\mathit{PEAS},\mathsf{PPBox},A} : w = 1] = \text{negl}$$

$$|\Pr[\mathsf{CSORAANON}^0_{\mathit{PEAS},\mathsf{PPBox},A} : g = 0] - \Pr[\mathsf{CSORAANON}^1_{\mathit{PEAS},\mathsf{PPBox},A} : g = 0]| = \text{negl} \ .$$

Before proving CSO-RA-ANON security we prove a technical lemma for a common proof pattern.

Parameters 1. Let $c > 0$ be a constant, called the honesty gap.
2. Let $M = (3+c)T$ be the number of parties, where T is the corruption budget of A.
3. Let $n = \lambda$ and let $\phi = \ell n$ be the largest multiple of n such that $\phi \leq (c/2)M$.

Init: Initially generate $(\mathsf{N}, \mathsf{sp}) \leftarrow \mathsf{Gen}$. Let $\mathcal{A}$ be the privacy structure of PPBox. Initialize PPBox by giving it $(\mathsf{N}, \mathsf{sp})$.

Run as RA Create an instance of $\mathcal{F}_{\text{RA+MPC}} = \mathcal{F}_{\text{RA+MPC}}^{\mathsf{PEAS},\epsilon=\frac{1}{3+c},1}$. We describe the game for $m = 1$ but it can be played for any adversarially chosen m, i.e., $\mathcal{F}_{\text{RA+MPC}} = \mathcal{F}_{\text{RA+MPC}}^{\mathsf{PEAS},\epsilon=\frac{1}{3+c},m}$. Let A be a PPT adversary interacting with $\mathcal{F}_{\text{RA+MPC}}$ as the environment, with the following exceptions.
- Do not give sp to A.
- When A inputs $(\text{DELIVERBATCH}, P)$ to $\mathcal{F}_{\text{RA+MPC}}$ for an honest P, then we do not give $(\mathsf{gsk}_P^{\mathsf{bo}_P}, \cdot)$ to A. Instead we save $(P, \mathsf{bo}_P, \mathsf{gsk}_P^{\mathsf{bo}_P})$ for later "execution".
- A may corrupt at most T parties.

Form Committees: When A inputs $(\text{NEXTBATCH}, \ldots)$ to $\mathcal{F}_{\text{RA+MPC}}$ run that command then then:
1. Let $(\mathsf{rpk}_{1,1}^b, \ldots, \mathsf{rpk}_{1,n}^b, \ldots, \mathsf{rpk}_{\ell,1}^b, \ldots, \mathsf{rpk}_{\ell,n}^b) = (\mathsf{rpk}_1^b, \ldots, \mathsf{rpk}_\phi^b)$.
2. Call $(\mathsf{rpk}_{j,1}^b, \ldots, \mathsf{rpk}_{j,n}^b)$ committee j of batch b.
3. Let r^b be the random tape used by $\mathcal{F}_{\text{RA+MPC}}$ to generate $\mathsf{gpk}^b, \mathsf{gpk}^b, \mathsf{rpk}^b$.

Run PPBox: On $(\text{PPBOX}, \mathsf{aux}, b, j)$ proceed as follows.
1. Input aux to PPBox.
2. Sample $(\mathsf{aux}', \mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{PPBox}$.
3. For $i = 1, \ldots, n$ sample $c_i \leftarrow \overline{\mathsf{Enc}}_{\mathsf{rpk}_{j,i}^b}(\mathsf{sh}_i)$.
4. Input $(\mathsf{aux}', c_1, \ldots, c_n)$ to A.

Execute Role If at any point A outputs $(\text{EXEC}, b, R \in \mathbb{P}^{\mathsf{b}})$, where for $P = \pi_b^{-1}(R)$ the value (P, b, gsk_P^b) is saved, then give (P, b, gsk_P^b) to the adversary.

Base IND-CPA: On input $(\text{ENCBASE}, f : \{0,1\}^* \to \mathbb{Z}_N^a)$ do:
1. Compute $(m_1^1, \ldots, m_a^1) = f(\mathsf{N}, \mathsf{sp}, r^1, \ldots, r^b)$.
2. Let $(m_1^0, \ldots, m_a^0) = (0, \ldots, 0)$.
3. Input $(c_1, \ldots, c_a) \leftarrow (\mathsf{Enc}_{\mathsf{N}}(m_1^d), \ldots, \mathsf{Enc}_{\mathsf{N}}(m_a^d))$ to the adversary.

Role PE Corruption If at any point A corrupts P then for each $1 \leq b \leq \mathcal{F}_{\text{RA+MPC}}.\mathsf{b}$ let $R_P^b = \pi_b(P)$. If the adversary has not previously given the command (EXEC, b, R_P^b) then we say that rpk_P^b was PE corrupted.

End of Game If there exist $0 \leq b \leq \mathcal{F}_{\text{RA+MPC}}.\mathsf{b}$ and $1 \leq j \leq \ell$ such that $\geq n/2$ roles in $(\mathsf{rpk}_{j,1}, \ldots, \mathsf{rpk}_{j,n})$ are PE corrupted, then let $w = 1$. Else, let $w = 0$. In addition let A output a guess $g \in \{0,1\}$.

Fig. 15. CSO-RA-ANON Game $\mathsf{CSORAANON}_{\mathsf{PEAS},A}^d$

Lemma 1 (Everywhere from Anywhere). *Consider two processes D^0 and D^1 which each run through a number of steps. Let A be a PPT adversary being shown a view of $D = D^b$ and having to guess b. Let Ab and Ba be events defined on both processes. Think of $\mathsf{Ba}(D^b)$ as something bad happening. We let $\mathsf{Ab}(D^b)$ (about to happen) denote the event that $\mathsf{Ba}(D^b)$ did not happen yet, but it will happen in the next step. Let $D^b_{\to\mathsf{Ab}}$ denote the process where we run D^b up until $\mathsf{Ab}(D^b)$ happens and then stops. So, $A(D^b_{\to\mathsf{Ab}})$ would have to make its guess using its view at this step just before Ba happens. Assume that $A(D^b)$ can detect $\mathsf{Ab}(D^b)$ in poly-time from its view in the game with D^b. Assume furthermore that we can prove the following for all PPT A:*

1. $|\Pr[A(D^0_{\to\mathsf{Ab}}) = 0] - \Pr[A(D^1_{\to\mathsf{Ab}}])| = \text{negl}$.
2. $\mathsf{Ba}(D^0)$ *happens with negligible probability in* $A(D^0)$.

Then for all PPT A

1. $|\Pr[A(D^0) = 0] - \Pr[A(D^1) = 0]| = \text{negl}$.
2. $\mathsf{Ba}(D^1)$ *happens with negligible probability in* $A(D^1)$.

Proof. We prove that last conclusion first. To see this observe that if $\Pr[\mathsf{Ba}(D^1)] \neq \text{negl}$ then because $\mathsf{Ba}(D^1)$ implies $\mathsf{Ab}(D^1)$ and A can detect $\mathsf{Ab}(D^1)$ in poly-time, we can consider the A outputting 1 when $\mathsf{Ab}(D^b)$ happens and outputs 0 if the execution ends without $\mathsf{Ab}(D^b)$ happening. From $\Pr[\mathsf{Ba}(D^0)] = \text{negl}$ we have that $\Pr[\mathsf{Ab}(D^0)] = \text{negl}$, so $\Pr[\mathsf{Ab}(D^0)] = \text{negl}$. Assume that for the sake of contradiction that it is not the case that $\mathsf{Ba}(D^1)$ happens with negligible probability in $A(D^1)$. From $\Pr[\mathsf{Ba}(D^1)] \neq \text{negl}$ we get that $\Pr[\mathsf{Ab}(D^1)] \neq \text{negl}$. But then $\Pr[\mathsf{Ab}(D^0)] = \text{negl}$ and $\Pr[\mathsf{Ab}(D^1)] \neq \text{negl}$. But by construction $\Pr[A(D^0)] = \Pr[A(D^0)_{\to\mathsf{Ab}}]$ and $\Pr[A(D^1)] = \Pr[A(D^1)_{\to\mathsf{Ab}}]$. This contradicts $|\Pr[A(D^0) = 0] - \Pr[A(D^1) = 0]| = \text{negl}$. Now since for all PPT A we have proven that $\mathsf{Ba}(D^1)$ happens with negligible probability in both $A(D^0)$ and $A(D^1)$ and we have assumed that that $|\Pr[A(D^0_{\to\mathsf{Ab}}) = 0] - \Pr[A(D^1_{\to\mathsf{Ab}})]| = \text{negl}$, we have that $|\Pr[A(D^0) = 0] - \Pr[A(D^1) = 0]| = \text{negl}$, as desired. □

Theorem 4. *If* PEAS *is SO-ANON secure and* $(\mathsf{PEAS}, \mathsf{PPBox})$ *is SO-IND-CPA secure and is CSO-IND-CPA secure, then* $(\mathsf{PEAS}, \mathsf{PPBox})$ *is CSO-RA-ANON secure.*

Proof. Consider an adversary A for $\mathsf{CSORAANON}^d_{\mathsf{PEAS},\mathsf{PPBox},A}$. Let

$$p^b = \Pr[\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A} : w = 1] \ .$$

We first argue that p^0 is negligible. We do this by constructing a PPT adversary B winning $\mathsf{RAANON}_{\mathsf{PEAS},B}$ with probability p^0. The adversary B will simulate $\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A}$. To do this reduction we use two main observations.

1. In $\mathsf{RAANON}_{\mathsf{PEAS},B}$ the adversary interacts with $\mathcal{F}_{\text{RA+MPC}}$ as the environment and therefore learns sp. The only restriction is that B is not given the secret global keys gsk^b_P until they are "executed".
2. The adversary B can still simulate **Base IND-CPA** in $\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A}$ as **Base IND-CPA** encrypts $(0, \ldots, 0)$. In particular, B does not need the secret global keys gsk^b_P to run **Base IND-CPA**.

It is easy to see that this allows B to simulate $\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A}$ in $\mathsf{RAANON}_{\mathsf{PEAS},B}$. In a bit more detail, B initialises PPBox using the $(\mathsf{N}, \mathsf{sp})$ from $\mathcal{F}_{\text{RA+MPC}}$. It can simulate **Run as RA** using its own access to **Run as RA**. It just does not pass sp to A. In **Form Committees** it will form the same committees, but will not get the random tape r^b used to sample the keys. It will run **Run PPBox** honestly. It maps **Execute Role** to **Execute Role** of $\mathsf{RAANON}_{\mathsf{PEAS},B}$. It simulates **Base IND-CPA** by encrypting $(0, \ldots, 0)$. Then note that the definitions of **Execute Role** are equivalent and if $w = 1$ in **End of Game** in $\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A}$ then B wins $\mathsf{RAANON}_{\mathsf{PEAS},B}$. This gives us

$$p^0 = \Pr[\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A} : w = 1] = \Pr[\mathsf{RAANON}_{\mathsf{PEAS},B} = 1] = \text{negl}(\lambda) \ .$$

We would now like to show that p^1 is negligible. We do this using Lemma 1. We let $D^b = \mathsf{CSORAANON}^b_{\mathsf{PEAS},\mathsf{PPBox},A}$. We let the bad event Ba be that $w = 1$, and we let Ab be

the event that the adversary has ordered a corruption which would lead to $w = 1$ if executed. We will first prove that

$$|\Pr[\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A\to\mathsf{Ab}} : g = 0] - \Pr[\mathsf{CSORAANON}^1_{\mathsf{PEAS},\mathsf{PPBox},A\to\mathsf{Ab}} : g = 0]| = \text{negl} \ . \tag{1}$$

We already know that $p^0 = \text{negl}$, i.e., $\Pr[\mathsf{Ba}(D^0)] = \text{negl}$. We use this to conclude that $p^1 = \Pr[\mathsf{Ba}(D^1)] = \text{negl}$ and in particular

$$|\Pr[\mathsf{CSORAANON}^0_{\mathsf{PEAS},\mathsf{PPBox},A} : g = 0] - \Pr[\mathsf{CSORAANON}^1_{\mathsf{PEAS},\mathsf{PPBox},A} : g = 0]| = \text{negl} \ .$$

We now prove Eq. (1) by showing that we can simulate $\mathsf{CSORAANON}^d_{\mathsf{PEAS},\mathsf{PPBox},A\to\mathsf{Ab}}$ from $\mathsf{SOINDCPA}^d_{\mathsf{PEAS},\mathsf{PPBox},C}$ for some PPT C. To see this, observe that as long as no committee has had too many PE corruptions it is easy to embed committees from $\mathsf{SOINDCPA}^d_{\mathsf{PEAS},\mathsf{PPBox},C}$ into $\mathsf{CSORAANON}^d_{\mathsf{PEAS},\mathsf{PPBox},A}$ by simply permuting the global keys of the committees obtained from $\mathsf{CSOINDCPA}^d_C$. If A makes a guess in $\mathsf{CSORAANON}^d_{\mathsf{PEAS},\mathsf{PPBox},A}$, let C do the same guess in $\mathsf{SOINDCPA}^d_{\mathsf{PEAS},\mathsf{PPBox},C}$. If A corrupts a party in $\mathsf{CSORAANON}^d_{\mathsf{PEAS},\mathsf{PPBox},A}$, let C corrupt the same party in $\mathsf{SOINDCPA}^d_C$. If A makes a corruption which under the permutation used for embedding would lead to too many PE corruptions on a committee, then C can detect this in PPT. And this is the Ab event, so the execution $\mathsf{CSORAANON}^d_{\mathsf{PEAS},\mathsf{PPBox},A\to\mathsf{Ab}}$ stops before C needs to execute the disallowed corruption. This gives us

$$\Pr[\mathsf{CSORAANON}^d_{\mathsf{PEAS},\mathsf{PPBox},A\to\mathsf{Ab}} = 0] = \Pr[\mathsf{SOINDCPA}^d_{\mathsf{PEAS},\mathsf{PPBox},C} = 0] \ ,$$

which via SO-IND-CPA gives us Eq. (1). □

C Justified ACS with Adjacent Output Rounds and Player Replaceable Committees

We adapt the protocol ACS protocol from [KN23] to the YOSO setting. The high-level changes are described in Section 7.2.

C.1 Sampling Committees using Cryptographic Sortition

Our TOB protocol does not rely on private channels and the committees can simply self-nominate using the cryptographic sortition implementation from [GHM+17], which is described in terms of parties who each have a weight w and where the sum of corrupted weight must be less than a third of the total weight W by some constant fraction. In short a party P_i who has w_i units of the total weight W computes $(h, \pi, j) = \mathsf{Sortition}(\mathsf{sk}_i, \mathsf{seed}, n, \mathsf{role}, w, W)$ to privately check how many parties they are emulating on the committee for role, and other parties can verify this using $\mathsf{VerifySort}(\mathsf{pk}_i, h, \pi, \mathsf{seed}, n, \mathsf{role}, w, W)$. Assuming the seed was chosen before the secret key of each user, the probability that each unit of weight gets to emulate a role is $\frac{n}{W}$. We present the protocol in the standard setting of M parties of equal weight, by simply fixing $w_i \coloneqq 1$ for all P_i and $W \coloneqq M$. Since the assignment of each party (or equivalently unit of weight) to a role is drawn independently from a Bernoulli distribution we can apply lemma 24 from [BKLZL20] which we restate (slightly simplified).

Lemma 2 (Lemma 24 from [BKLZL20]). *If $n \leq M$, $0 < \epsilon < 1/3$, and $T \leq (1-2\epsilon)M/3$ bounds the number of corruptions, then a committee C_{role} sampled as above satisfies the following except with probability negligible in n:*

1. *C_{role} contains fewer than $(1+\epsilon) \cdot n$ parties.*
2. *C_{role} contains more than $((1+\epsilon/2) \cdot 2 \cdot n)/3$ honest parties.*
3. *C_{role} contains fewer than $(1-\epsilon) \cdot n/3$ corrupted parties.*

In the following protocols we will be using $\mathsf{P} \in \mathsf{C}_{\mathsf{role}}$ as shorthand for a party P who was elected for the committee of role and who implicitly sends proofs of this along with messages. This does not affect communication complexity as the parties are already sending a $O(\lambda)$ bit hash. Additionally we use n as the expected size of $\mathsf{C}_{\mathsf{role}}$, and assume that $t \coloneqq (1-\epsilon) \cdot n/3$ is an upper bound on the number of corrupted parties on $\mathsf{C}_{\mathsf{role}}$. We can think of n as a statistical security parameter. Additionally, except for the committees used to implement RB, the committees only need to have honest majority, so we can have concretely smaller committees in those instances. For simplicity we let n be security parameter λ.

C.2 Eventual Justifiers

We will use the definition of Eventual Justifiers from [KN23]. An eventual justifier is a predicate evaluated on a message and a party's local view. They are required to be monotone and propagating. Monotone in the sense that a party seeing a message as justified in their view, does not at some later point consider it to not be justified. Propagating in the sense that a message being justified at one party means that it will eventually be justified at all honest parties. A recent example of justifiers being used in a synchronous model include [LN23].

Definition 6 (Justifier [KN23]). *For a message identifier mid we say that J^{mid} is a justifier if the following properties hold.*

Monotone: *If for an honest P and some time τ it holds that $J^{\mathsf{mid}}(m, \mathsf{P}, \tau) = \top$ then at all $\tau' \geq \tau$ it holds that $J^{\mathsf{mid}}(m, \mathsf{P}, \tau') = \top$.*

Propagating: *If for honest P and some point in time τ it holds that $J^{\mathsf{mid}}(m, \mathsf{P}, \tau) = \top$, then eventually the execution will reach a time τ' such that $J^{\mathsf{mid}}(m, \mathsf{P}', \tau') = \top$ for all honest parties P'.*

The justifiers are often used as an explanation for why a party sent a particular message. It is natural to implement this by considering the predicate satisfied when a subset of the messages you received would prompt you to send the same message if you were performing the same role as the sender. Most of the protocol definitions in this section will have justifiers on inputs and outputs, meaning that all inputs and outputs can be guaranteed to have certain properties. This holds even for adversarial inputs and outputs, in the sense that if they are accepted as justified in the view of an honest party, then they satisfy some predicate. We say in that case that all *possible justified outputs* of a protocol satisfy the predicate. Similarly, most protocol messages will come with justifiers which are used to reason about all *possible*

justified messages of some type satisfying some property. In many cases this means that an adversary might lie about what message they should have sent in the protocol, but it will still be a message that can be explained as something an honest party would have sent based on a valid sequence of events, and therefore a message that is as good as what an honest party would have sent. We refer to [KN23] for a formal definition of possible justified messages and outputs. Combining justifiers with RB means that Byzantine parties cannot equivocate *and* have to give a explanation for why they send each message, which in many cases can combine to make an adversarial message have all the relevant properties we require of an honest one.

C.3 Reliable Broadcast for long messages

We present a protocol for reliable broadcast (RB) it has the usual properties of Bracha's RB[Bra87].

Definition 7 (Reliable Broadcast). *A protocol for M parties $\mathsf{P}_1, \ldots, \mathsf{P}_M$, where all parties have input mid. The message identifier mid contains the identity of a sender P_s.*

Validity: *If honest P_s has input (mid, m) and an honest P_i has output (mid, m') then $m' = m$.*

Agreement: *For all honest outputs (mid, m) and (mid, m') it holds that $m = m'$.*

Eventual Output 1: *If P_s is honest and has input $(\mathsf{mid}, \cdot)$, and all honest P_j start running the protocol, then eventually all honest P_i have output $(\mathsf{mid}, \cdot)$.*

Eventual Output 2: *If an honest P_j has output $(\mathsf{mid}, \cdot)$, and all honest parties start running the protocol, then eventually all honest P_i have output $(\mathsf{mid}, \cdot)$.*

Our protocol for subquadratic and message length optimal RB follows the blueprint of algorithm 4 by Das et al. [DXR21], but relies on a distinct self-nominated committee to perform each activation rule. At a high level: after receiving the message from the designated sender the remaining parties essentially run Bracha's RB protocol on a hash of the message while distributing shares of a Reed-Solomon encoding of the message. Each party P_i is supposed to receive the i^{th} share of the encoding from each party sending an echo message. So if all honest parties receive the same message from the sender, then they forward a matching hash and shares to each other party. It follows that if any message is supported by the echo messages of a supermajority (making it unique), then any honest party P_i who sends a ready message includes the i^{th} share of that message. We restate it in Fig. 16 to illustrate how it can have subquadratic communication when $T < (1-\epsilon)M/3$ by instantiating it with committees.

We only need to observe that it is YOSO (i.e., each committee member sends only one message) and that when $T < (1-\epsilon)M/3$ we can by Lemma 2 sample committees of size $O(\lambda)$ where at most n parties are corrupted and more than twice as many are honest. Then lowering the degrees of the polynomials used in the Reed-Solomon code to t means that from any set of distinct shares greater than $3t$ of which at most t shares are incorrect one can reconstruct the message. To be able to send a distinct share to all M parties we need

Protocol Π_{RB}

Send: P_s sends m to all parties.

Echo: On receiving the first valid message m_i from P_s each party $\mathsf{P}_i \in \mathsf{C}_{\mathsf{echo}}$ computes the Reed-Solomon encoding and hash of their value $\mathsf{D}_i = (s_1, \ldots, s_n) = \mathsf{Encode}(m_i)$, $h_i = \mathsf{H}(m_i)$, and sends $(\mathsf{echo}, s_j, h_i)$ to each party $\mathsf{P}_j \in \mathcal{P}$.

Ready 1: On receiving messages of the form $(\mathsf{echo}, s_i^1, h)$ with the same values of s_i^1 and h from $n-t$ distinct parties in $\mathsf{C}_{\mathsf{echo}}$ each party $\mathsf{P}_i \in \mathsf{C}_{\mathsf{ready}}$ who has not yet sent a ready message sends $(\mathsf{ready}, s_i^1, h)$ to all parties.

Ready 2: On receiving messages of the form $(\mathsf{ready}, \cdot, h)$ from $t+1$ distinct parties in $\mathsf{C}_{\mathsf{ready}}$ and messages of the form $(\mathsf{echo}, s_i^2, h)$ from $t+1$ distinct parties $\mathsf{C}_{\mathsf{echo}}$ with the same hash h and share s_i^2, each party $\mathsf{P}_i \in \mathsf{C}_{\mathsf{ready}}$ who has not yet sent a ready message sends $(\mathsf{ready}, s_i^2, h)$ to all parties.

Output: On receiving messages of the form $(\mathsf{ready}, s'_j, h)$ from at least $n-t$ distinct parties each party P tries to reconstruct from the shares received. To reconstruct, P first removes any shares from parties sending more than one share and then, if reconstruction using the remaining shares is successful, outputs the result and terminates. This step is repeated each time a new ready message is received, until successful reconstruction.

Fig. 16. Reliable Broadcast

to pick the polynomials of the Reed-Solomon code to be over a field that is larger than M. Concretely the messages just need to be of size $\log(M) \cdot \lambda$ to dominate the total combined sized of the points sent by the committee, resulting in message length optimality, but since all parties will be including a cryptographic hash, a signature, and the output of a VRF from sortition in their messages, the dominating cost will be λ parties sending λ bits to M parties. This lowers the communication complexity of broadcasting an $|m|$ bit message from $O(M(|m| + M\lambda))$ in [DXR21] to $O(M(|m| + \lambda^2))$.

The main insight is that if two committees with honest supermajority perform the echo and ready roles, and the reconstruction threshold of the Reed-Solomon code is less than a third of the committee size, the proof still goes through: If an honest party, P, terminates because they saw $n - t$ ready messages, then (as in Bracha's original protocol) $t + 1$ honest parties sent ready, and at least one of them, P', did so because they saw $n - t$ echo messages. At least $t + 1$ of matching echo messages seen by P' came from honest parties, and the messages of these $t + 1$ honest parties will eventually arrive at the remaining honest parties (with a different set of matching shares). So now every honest party will eventually see $t + 1$ echo messages (from the parties who sent echo to P') with the same hash and same share, $t + 1$ ready messages (from the honest parties who sent ready to P) with the same hash, allowing them to reconstruct the message from the shares in the consistent ready messages, and send their own share in a ready message. All honest parties sending a ready message in turn allows every honest party to terminate.

C.4 Causal Cast

The concept of Causal Cast (CC)[KN23] is an abstraction over DAG based protocols that utilize the structure of a DAG to infer what a party would have said in a protocol that they are in an abstract sense running without directly sending the messages. It can be thought of

as a tool to describe protocols in this paradigm (notable examples include [Bai16,KKNS21]) without needing to explicitly consider the structure of the DAG. Using the terminology of CC a message of a party in the protocol is a *computed message* when it can be inferred by pointing to previous messages instead of explicitly sending the message. This concept was pioneered and dubbed "virtual voting" in [Bai16]. If a message, such as a message in a block, needs to be introduced to the DAG, then it is instead a *free-choice message*. For motivation of the remaining concepts we refer to [KN23]. What is important for our purposes is that CC is used black box in [KN23] and that we can implement it in the YOSO model by giving YOSO implementations of Reliable Broadcast and $\mathcal{F}_{\text{CF}}$, which are in turn used black box to implement CC in [KN23]. We provide a YOSO RB in Appendix C.3 and assume access to an ideal coin functionality $\mathcal{F}_{\text{CF}}$. However, there is the caveat that the ideal coin functionality does not immediately provide an implementation of Leader Election, because the committees are sampled using sortition. We show how to get around this hurdle in Appendix C.7.

Definition 8 (Causal Cast[KN23]). *A protocol for M parties $\mathsf{P}_1, \ldots, \mathsf{P}_M$ is called a Causal Cast (CC) if it has the following properties.*

Free-Choice Send: *A party P_i can have input $(\text{CC-SEND}, \mathsf{mid}, m)$ where mid is a free choice identifier $\mathsf{P}_i = \mathsf{P}^{\mathsf{mid}}$ and $J_{IN}^{\mathsf{mid}}(m) = \top$ at $\mathsf{P}^{\mathsf{mid}}$ at the time of input.*

Computed-Message Send: *A party P_i can have input $(\text{CC-SEND}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$, where mid is a computed-message identifier, $\mathsf{P}_i = \mathsf{P}^{\mathsf{mid}}$, P_i earlier gave outputs $(\text{CC-DEL}, \mathsf{mid}_j, m_j)$ for $j = 1, \ldots, \ell$, and*

$$\bot \neq m = \mathsf{NextMessage}^{\mathsf{mid}}((\mathsf{mid}_1, m_1), \ldots, (\mathsf{mid}_\ell, m_\ell)) \ .$$

Constant Send: *A party P_i can have input $(\text{CC-SEND}, \mathsf{mid}, m)$ where mid is a constant identifier. In that case it is guaranteed that all parties eventually have the same input $(\text{CC-SEND}, \mathsf{mid}, m)$.*

Free-Choice Validity: *A party P_i can have output $(\text{CC-DEL}, \mathsf{mid}, m)$, where mid is a free-choice identifier. It then holds that $J_{IN}^{\mathsf{mid}}(m) = \top$ at P_i at the time of output. Furthermore, if $\mathsf{P}_j = \mathsf{P}^{\mathsf{mid}}$ is honest, then P_j had input $(\text{CC-SEND}, \mathsf{mid}, m)$.*

Coin Flip Validity: *A party P_i may output $(\text{CC-DEL}, \mathsf{mid}, m)$ where mid is a coin-flip identifier mapping to an instance of $\mathcal{F}_{CF}$ as defined in Fig. 29 and the index of a coin: ℓ. In that case P_i has previously received output $\mathcal{F}_{CF}.L[\ell]$ from $\mathcal{F}_{CF}$.*

Computed-Message Validity: *A party P_i can have output $(\text{CC-DEL}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$, where mid is a computed-message identifier. In that case P_i earlier gave outputs $(\text{CC-DEL}, \mathsf{mid}_j, m_j, \ldots)$ for $j = 1, \ldots, \ell$, and $\bot \neq m = \mathsf{NextMessage}^{\mathsf{mid}}((\mathsf{mid}_1, m_1), \ldots, (\mathsf{mid}_\ell, m_\ell))$.*

Constant Validity: *A party P_i can have output $(\text{CC-DEL}, \mathsf{mid}, m)$. In that case it immediately before had input $(\text{CC-SEND}, \mathsf{mid}, m)$.*

Liveness: *If an honest party P_i had input $(\text{CC-SEND}, \mathsf{mid}, \ldots)$ or some honest party had output $(\text{CC-DEL}, \mathsf{mid}, \ldots)$ and all honest parties are running the system, then eventually all honest parties have output $(\text{CC-DEL}, \mathsf{mid}, \ldots)$.*

Agreement: *For all possible justified outputs $(\text{CC-DEL}, \mathsf{mid}, m, \ldots)$ and $(\text{CC-DEL}, \mathsf{mid}, m', \ldots)$ it holds that $m' = m$.*

We will also be using the notion of Justified Causal Cast protocols ([KN23]) in which outputs are associated with a message identifier mid and can be sent as a computed message $(\text{CC-SEND}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$, in which case the message identifiers $\mathsf{mid}_1, \ldots, \mathsf{mid}_\ell$ justify the output. For a Justified Causal Cast protocol Π we will use $\Pi.J_{\text{OUT}}$ to denote its output justifier. This will be useful for reporting justified outputs of subprotocols.

Remark 1 (Honest Majority Committees). For the remaining protocols in this section we only need "honest majority" committees, by which we mean that the following holds except with probability negligible in n:

1. $\mathsf{C}_{\mathsf{role}}$ contains fewer than $(1+\epsilon)\cdot n$ parties.
2. $\mathsf{C}_{\mathsf{role}}$ contains more than $((1+\epsilon)\cdot n)/2$ honest parties.
3. $\mathsf{C}_{\mathsf{role}}$ contains fewer than $(1-\epsilon)\cdot n/2$ corrupted parties.

This is implied by the bounds in Lemma 2, but in practice allows sampling committees that are concretely smaller by picking an appropriately smaller n and letting $t \coloneqq (1-\epsilon)\cdot n/2$.

C.5 Justified Gather

We restate the definition Justified Gather (Definition 9) and the protocol Π_{GATHER} (Fig. 17) which implements it. Nothing changes from [KN23] apart from notation and each activation rule being performed by parties who self-nominate using sortition as described in Appendix C.1.

Definition 9 (Justified Gather). *A protocol for M parties $\mathsf{P}_1, \ldots, \mathsf{P}_M$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at P_i at the time the input is given.*

Liveness: *If all honest parties start running the protocol with a J_{IN}-justified input then eventually all honest parties have a J_{OUT}-justified output.*

Justified Blocks: *For all possible justified outputs U and all (potentially corrupt) P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that $J_{IN}(B_i) = \top$.*

Validity: *For all possible justified outputs U and all honest P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that P_i had input B_i.*

Agreement: *For all possible justified outputs U and U' and all $(\mathsf{P}_i, B_i) \in U$ and $(\mathsf{P}_i, B_i') \in U'$ it holds that $B_i = B_i'$.*

Large Core: *For all possible justified outputs $(U^1, \ldots, U^m)$ it holds that $|\bigcap_{k=1}^m U^k| \geq n - t$.*

The proof that Π_{GATHER} is a Justified Gather protocol is presented in [KN23]. The only change is that the dimensions of the table T and the combinatorial argument changes from using a fixed committee size n_{C} and corruption bound t_{C} with $t_{\mathsf{C}} < n_{\mathsf{C}}/2$ to reasoning about committees of random but bounded size elected using sortition. Concretely, let $n_{\mathsf{Gather;2}}$ be the number of parties in $\mathsf{C}_{\mathsf{Gather;2}}$ and $n_{\mathsf{Gather;3}}$ be the number of parties in $\mathsf{C}_{\mathsf{Gather;3}}$, then the table will have $n_{\mathsf{Gather;2}}$ rows and $n_{\mathsf{Gather;3}}$ columns. But this does not change the conclusion, as in either case the $n-t$ sets included in the unions will (except with negligible probability) be more than half than the maximal actual committee size for all other committees in the protocol. This holds even with the bounds in Remark 1.

1. The input of P_i is B_i with $J_{\text{IN}}(B_i) = \top$. Party P_i lets $U_i^0 = \{(\mathsf{P}_i, B_i)\}$. The singleton set is justified by B_i satisfying J_{IN}.
2. For $r \in [1;3]$ each party $\mathsf{P}_i \in \mathsf{C}_{\mathsf{Gather;r\text{-}1}}$ causal casts U_i^{r-1} as a computed message. Then each party P_i collects incoming U_j^{r-1} from parties $\mathsf{P}_j \in \mathsf{C}_{\mathsf{Gather;r\text{-}1}}$, lets P_i^r be the set of P_j it heard from, waits until $|P_i^r| \geq n - t$ and lets
$$U_i^r = \bigcup_{\mathsf{P}_j \in P_i^r} U_j^{r-1}.$$
The message is justified by being computed from the set P_i^r where $|P_i^r| \geq n - t$.
3. Finally, P_i outputs U_i^3.

Fig. 17. Protocol Π_{Gather}.

C.6 Justified Graded Gather

We restate the definition of a Justified Graded Gather protocol from [KN23] in Definition 10 and the protocol, $\Pi_{\text{GradedGather}}$ implementing it in Fig. 18.

Definition 10 (Justified Graded Gather). *A protocol for M parties $\mathsf{P}_1, \ldots, \mathsf{P}_M$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at P_i at the time the input is given.*

Liveness: *If all honest parties start running the protocol with a J_{IN}-justified input then eventually all honest parties have a J_{OUT}-justified output.*

Justified Blocks: *For all possible justified outputs (U, T) and all (potentially corrupt) P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that $J_{IN}(B_i) = \top$.*

Sub Core: *For all possible justified outputs $((U^1, T^1), \ldots, (U^m, T^m))$ it holds that $T^i \subseteq \bigcap_{k=1}^m U^k$ for all $i \in [m]$.*

Validity: *For all possible justified outputs (U, T) and all honest P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that P_i had input B_i.*

Agreement: *For all possible justified outputs (U, T) and (U', T') and all $(\mathsf{P}_i, B_i) \in S$ and $(\mathsf{P}_i, B_i') \in U'$ it holds that $B_i = B_i'$.*

Large Sub Core: *For all possible justified outputs $((U^1, T^1), \ldots, (U^m, T^m))$ it holds that $|\bigcap_{k=1}^m T^k| \geq n - t$.*

As in Appendix C.5 besides the committee being self-nominating nothing substantial changes, and the proof follows from the one in [KN23] because the bounds in Remark 1 imply intersection between any two subsets of the committee of size $n - t$.

C.7 Justified Graded Block Selection

Justified Graded Block Selection as defined in [KN23] allows a set of parties to input a justified block and get one as output alongside a grade with the following properties:

Definition 11 (Justified Graded Block Selection[KN23]). *A protocol for M parties $\mathsf{P}_1, \ldots, \mathsf{P}_M$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at the time the input B_i is given. The output of the protocol is a block C_i justified by J_{OUT}.*

1. The input of P_i is B_i with $J_{\text{IN}}(B_i) = \top$. All parties run Π_{GATHER} with P_i inputting B_i justified by J_{IN}. Let the output of P_i be U_i'. If $\mathsf{P}_i \in \mathsf{C}_{\mathsf{GradedGather}}$ it then causal casts U_i' as a computed-message justified by $\Pi_{\text{GATHER}}.J_{\text{OUT}}$.
2. Party P_i collects U_j' from parties $\mathsf{P}_j \in \mathsf{C}_{\mathsf{GradedGather}}$, lets P_i be the set of P_j it heard from and waits until $|P_i| \geq n - t$.
3. Party P_i outputs
$$(U_i, T_i) = \big(\bigcup_{\mathsf{P}_j \in P_i} U_j' \ , \quad \bigcap_{\mathsf{P}_j \in P_i} U_j' \big).$$
The outputs are justified by being computed as above from justified sets.

Fig. 18. $\Pi_{\text{GRADEDGATHER}}$

Liveness: *If all honest parties start running the protocol with a J_{IN}-justified input then eventually all honest parties have a J_{OUT}-justified output.*

Justified Output: $J_{\text{IN}}(C_i) = \top$ *holds for all possible J_{OUT}-justified outputs C_i.*

Graded Agreement: *For all possible justified outputs (C_i, g_i) and (C_j, g_j) it holds that $|g_i - g_j| \leq 1$. Furthermore, if both $g_i, g_j > 0$ then $C_i = C_j$.*

Positive Agreement: *There exists $\alpha > 0$ such that with probability at least $\alpha - \text{negl}$ all possible justified outputs of at least $n - t$ parties will have grade $g_i = 2$.*

Stability: *If there are possible justified outputs (C_i, g_i) and (C_j, g_j) with $C_i \neq C_j$ then there exist two justified inputs B_i and B_j with $B_i \neq B_j$.*

We will start out by presenting a protocol, $\Pi_{\mathsf{WeakGradedSelectBlock}}$, with a weakened version of the Positive agreement property:

Weak Positive Agreement There exists $\alpha > 0$ such that with probability at least $\alpha - \text{negl}$ some honest P_i will have output (C_i, g_i) with $g_i = 2$.

This will in turn be used to implement a full fledged Justified Graded Block Selection protocol with a strengthened Stability property in Appendix C.8.

$\Pi_{\mathsf{WeakGradedSelectBlock}}$ is presented in Fig. 19. It is using the core principles from the corresponding primitive in [KN23] but needs a few modifications to function in our setting where the committees are self-nominated. The main challenge is that parties do not have a description of the committee. They do not even know its exact size. So, we cannot in a straightforward manner map a random string to a member of the committee. Instead, for each committee member seen in our accumulated set we will feed the coin output together with their party identifier through the random oracle, H, and obtain a string which was unpredictable before the sub core of the accumulated sets were fixed for the first honest party. We will locally regard the party who has the lexicographically least string as a leader candidate and then gossip candidates to get graded agreement on a leader and their block. If it happens that the committee member with the least string is in a sub core, T_i, then P_i gets a grade 2 output. Due to the Large Sub Core property of $\Pi_{\text{GRADEDGATHER}}$ this happens with good constant probability as at least one honest T_i is fixed when the first honest party initiates $\mathcal{F}_{\text{CF}}$.

When instantiating this $\Pi_{\mathsf{WeakGradedSelectBlock}}$ as shown in Fig. 19 with $\mathcal{F}_{\text{CF}}$ from Fig. 29, then it satisfies Definition 11 with Weak Positive Agreement.

1. The input of P_i is B_i with $J_{\text{IN}}(B_i) = \top$.
2. The parties run $\Pi_{\text{GRADEDGATHER}}$ with input B_i and input justifier J_{IN}. Let the output of P_i be (U_i, T_i).
3. After getting output from $\Pi_{\text{GRADEDGATHER}}$ the parties input NEXT-COIN to $\mathcal{F}_{\text{CF}}$ and then COIN-INDEX to get the corresponding coin identifier ℓ.
4. On output (ℓ, coin) from $\mathcal{F}_{\text{CF}}$ if $\mathsf{P}_i \in \mathsf{C}_{\mathsf{FirstRoundCandidate}}$: for each $\mathsf{P}_j \in U_i$ let $\mathit{ticket}_j = \mathsf{H}(\mathsf{P}_j \| \mathit{coin})$, let P_k be the one with the lexicographically least ticket_k value, and send $(\mathsf{FirstRoundCandidate}, \mathsf{P}_k)$ as a Causal Cast message computed from the set U_i and coin.
5. On receiving $n - t$ $(\mathsf{FirstRoundCandidate}, \mathsf{P}_k)$ messages from parties in $\mathsf{C}_{\mathsf{FirstRoundCandidate}}$ each $\mathsf{P}_i \in \mathsf{C}_{\mathsf{SecondRoundCandidate}}$: If all the relayed P_k are identical lets $b_i = \top$, and otherwise $b_i = \bot$ and finally sends $(\mathsf{SecondRoundCandidate}, \mathsf{P}_k)$ as a Causal Cast computed message based on the set of received $\mathsf{FirstRoundCandidate}$ messages.
6. On receiving $n-t$ $(\mathsf{SecondRoundCandidate}, b_j)$ messages from parties in $\mathsf{C}_{\mathsf{SecondRoundCandidate}}$ each P_i: lets n_i be the number of messages where $b_j = \top$, if $n_i > 0$ lets P_k be a party included in $n-t$ $\mathsf{FirstRoundCandidate}$ messages, and outputs

$$(C_i, g_i) = \begin{cases} (B_k, 2) & \text{if } n_i \geq n - t \wedge \exists(\mathsf{P}_k, B_k) \in T_i \\ (B_k, 1) & \text{if } n_i > 0 \wedge \exists(\mathsf{P}_k, B_k) \in U_i \setminus T_i \\ (B_i, 0) & \text{if } \nexists(\mathsf{P}_k, \cdot) \in U_i \vee n_i = 0\ . \end{cases}$$

   and if it is on the committee $\mathsf{C}_{\mathsf{GradedSelectBlock}}$ Causal Casts (C_i, g_i). The output is justified by being computed as above from justified values.

Fig. 19. $\Pi_{\mathsf{WeakGradedSelectBlock}}$

Lemma 3. *$\Pi_{\mathsf{WeakGradedSelectBlock}}$ satisfies Justified Graded Block Selection Definition 11 with Weak Positive Agreement.*

Proof. Liveness and Justified Output are trivial. Stability holds because the output justification transitively refers to a justified input through computed messages. We argue Graded Agreement: Assume a party P_i has output $(B_k, 2)$. Then $n_i \geq n-t$ and $B_k \in T_i$. Now for any other party P_j the Sub Core property of $\Pi_{\text{GRADEDGATHER}}$ ensures that $B_k \in U_j$, and because all second round messages were sent through CC (i.e. without equivocations) $n_i \geq n-t$ implies $n_j \geq n-2t > 0$. So, we have taken care of $|g_i - g_j| \leq 1$ as P_j must now have grade at least 1. (If no party has grade 2 there is nothing to show.) Now consider any party P_i with output grade at least 1. This party had $n_i > 0$, and thus received at least one $(\mathsf{SecondRoundCandidate}, \top)$ message justified by $n-t$ $(\mathsf{FirstRoundCandidate}, \mathsf{P}_k)$ messages on the same party P_k which by intersection and the messages being sent through CC implies that no other party P_l can be included in $n-t$ $\mathsf{FirstRoundCandidate}$ messages, which in turn means that only B_k can get a grade of 1 or 2. We finally argue Weak Positive Agreement: When modelling H as a random oracle: except with negligible probability there are no collisions among the outputs of H, and the probability that the party, P_i, which has the lexicographically least ticket_i value globally in a set S, is in any subset of size $c|S|$ where the subset is independent from coin is $c - \text{negl}$. Let P_j be the first honest party to give input NEXT-COIN to $\mathcal{F}_{\text{CF}}$. In particular P_j already had output (T_j, U_j) from $\Pi_{\text{GRADEDGATHER}}$ while the value of coin was unpredictable, so an adversary cannot have chosen T_j to correlate with coin. By the Large Sub Core property of $\Pi_{\text{GRADEDGATHER}}$ the intersection of all possible justified T values has size at least $n - t$. In particular, the probability that P_i has the lexicographically least ticket_i value is in T_j is

at least $\frac{n-t}{t}$ − negl.[6] Assume $\mathsf{P}_i \in T_j$. Then by Sub Core $\mathsf{P}_i \in U_k$ for all possible justified U_k and thus $(\mathsf{FirstRoundCandidate}, \mathsf{P}_i)$ is the only justifiable $\mathsf{FirstRoundCandidate}$ message, which means that $(\mathsf{SecondRoundCandidate}, \top)$ is the only justifiable $\mathsf{SecondRoundCandidate}$ message and the output of P_j must be $(B_i, 2)$.

C.8 Justified Strongly Stable Graded Block Selection

For our construction it will be useful to make sure that when instances of a Justified Graded Block Selection are run sequentially with the outputs being fed back as justified inputs to the next iteration, then whenever a party gets an output with grade 2 all other parties receive grade 2 in the next iteration. This is ensured by adding the following Strong Stability property.

Definition 12 (Justified Strongly Stable Graded Block Selection). *A Justified Graded Block Selection protocol that additionally satisfies the following Strong Stability property.*

Strong Stability: *If there is a possible justified output (C_i, g_i) with $g_i < 2$ then there exist two justified inputs B_i and B_j with $B_i \neq B_j$.*

Given a protocol $\Pi_{\mathsf{WeakGradedSelectBlock}}$ satisfying Definition 11 with Weak Positive Agreement we construct $\Pi_{\mathsf{StronglyStableGradedSelectBlock}}$ by adding two rounds of Causal Cast in Fig. 20 and show that it satisfies Definition 12.

Lemma 4 (Strong Stability). *$\Pi_{\mathsf{StronglyStableGradedSelectBlock}}$ is a Justified Strongly Stable Graded Block Selection protocol.*

Proof. Liveness and Justified Output is trivial. To get soft grade $h = 1$ one has to see the same block from a majority. Since all block are sent through RB, at most one block can have votes from a majority. It follows that blocks with grade $g > 0$ are identical as they justified by at least one block with soft grade $h = 1$. It is impossible for one party to get grade $g = 0$ and another to get grade $g = 2$ as each requires a majority of votes on soft grades $h = 0$ and $h = 1$ respectively. Finally the strong stability follows from the input values being justified, so if only one block is justified by J_{IN}, then all parties get $h = 1$ and $g = 2$. Note that the Justified output property means that the inner protocol $\Pi_{\mathsf{WeakGradedSelectBlock}}$ preserves the input justifier. For the same reason positive agreement holds, in fact a stronger statement holds: a single party getting grade $g = 2$ in the $\Pi_{\mathsf{WeakGradedSelectBlock}}$ results in everyone getting grade $g = 2$.

[6] Note that while we informally refer to the committees as having honest majority, Remark 1 in fact specifies that the honest parties outnumber the adversary by a number which is a constant fraction of the committee size, so the probability of terminating is constant in every round.

Protocol $\Pi_{\mathsf{StronglyStableGradedSelectBlock}}$

1. The input of P_i is B_i with $J_{\text{IN}}(B_i) = \top$.
2. Party P_i runs $\Pi_{\mathsf{WeakGradedSelectBlock}}$ with B_i as input using $J_{\text{IN}}(B_i)$ as justifier, and gets output $(B_i^1, \cdot)$. If $\mathsf{P}_i \in \mathsf{C}_{\mathsf{Upgrade1}}$ it causal casts $(\mathsf{Upgrade1}, B_i^1)$ justified by $\Pi_{\mathsf{WeakGradedSelectBlock}}.J_{\text{OUT}}$.
3. Party P_i collects justified messages $(\mathsf{Upgrade1}, B_j^1)$ from at least $n-t$ parties $\mathsf{P}_j \in P_i^1 \subseteq \mathsf{C}_{\mathsf{Upgrade1}}$ and lets

$$(B_i^1, h_i) = \begin{cases} (B, 1) & \text{if } \exists B : |\{\mathsf{P} \in P_i^1 | \mathsf{P} \text{ sent } (\mathsf{Upgrade1}, B)\}| \geq n - t \\ (\perp, 0) & \text{otherwise.} \end{cases}$$

   If $\mathsf{P}_i \in \mathsf{C}_{\mathsf{Upgrade2}}$ it causal casts $(\mathsf{Upgrade2}, B_i^2, h_i)$ justified by P_i^1.
4. Party P_i collects justified messages $(\mathsf{Upgrade2}, B_j^2, h_j)$ from at least $n-t$ parties $\mathsf{P}_j \in P_i^2 \subseteq \mathsf{C}_{\mathsf{Upgrade2}}$.

$$(C_i, g_i) = \begin{cases} (B, 2) & \text{if } \forall \mathsf{P} \in P_i^2 : \mathsf{P} \text{ sent } (\mathsf{Upgrade2}, B, 1) \\ (B, 1) & \text{if } \exists \mathsf{P} \in P_i^2 : \mathsf{P} \text{ sent } (\mathsf{Upgrade2}, B, 1) \\ (B_i, 0) & \text{if } \forall \mathsf{P} \in P_i^2 : \mathsf{P} \text{ sent } (\mathsf{Upgrade2}, \perp, 0) \ . \end{cases}$$

   Output (C_i, g_i) with output justifier P_i^2.

Fig. 20. $\Pi_{\mathsf{StronglyStableGradedSelectBlock}}$

C.9 Justified Block Selection with Adjacent Output Round Agreement

We now present a modified version of the Justified Block Selection primitive from [KN23]. It satisfies all of the properties of the original primitive, but adds a round number to the output and the guarantee that all parties give output in adjacent rounds. As in the original protocol the parties repeatedly execute a Justified Graded Block Selection protocol until a block is selected with grade 2 which guarantees that all other parties selected the same block with at at least grade 1. Because we are using a version with Strong Stability, all honest parties will output in adjacent rounds, and moreover it is impossible to cook up a justification for outputting in a round where an honest party could not have given output.

Definition 13 (Justified Block Selection with Adjacent Output Round Agreement). *A protocol for M parties $\mathsf{P}_1, \ldots, \mathsf{P}_M$. There is an input justifier J_{IN} and an output justifier J_{OUT}. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at the time the input was given. The output of the protocol is a block (C_i, r_i) justified by J_{OUT}.*

Liveness: *If all honest parties start running the protocol with a J_{IN}-justified input then eventually all honest parties have a J_{OUT}-justified output.*
Justified Output: *$J_{IN}(C_i) = \top$ holds for all possible J_{OUT}-justified outputs C_i.*
Agreement: *For all possible justified outputs (C_i, r_i) and (C_j, r_j) it holds that $C_i = C_j$.*
Adjacent Output Round Agreement: *For all possible justified outputs (C_i, r_i) and (C_j, r_j) it holds that $|r_i - r_j| \leq 1$, and if $(\cdot, r)$ is a justified output then no honest party sent a message in any round $r' > r + 1$.*

The protocol $\Pi_{\mathsf{SelectBlock}}$ is identical to the protocol in [KN23] except it is adapted to use committees, it has a round number added to its output, and it uses Strongly Stable Graded

Protocol $\Pi_{\mathsf{SelectBlock}}$

1. Each party P_i initialises $\mathsf{GaveOutput}_i = \bot$.
2. Each party $\mathsf{P}_i \in \mathsf{C}_{\mathsf{SelectBlockInput}}$ with input B_i where $J_{\text{IN}}(B_i) = \top$, lets $B_i^0 = B_i$ and $g_i^0 = 0$ and Causal Casts (B_i^0, g_i^0), which is justified by $J_{\text{IN}}(B_i^0) = \top$ and $g_i^0 = 0$.
3. For rounds $r = 1, \ldots$ each part P_i with $\mathsf{GaveOutput}_i = \bot$ runs $\Pi_{\mathsf{StronglyStableGradedSelectBlock}}$ where:
   (a) P_i has input B_i^{r-1}.
   (b) The input of P_i is justified by a justified (B_i^{r-1}, g_i^{r-1}) with $g_i^{r-1} < 2$.
   (c) P_i eventually gets justified output (B_i^r, g_i^r).
4. In addition to the above loop each P_i runs the following *echo rules*:
   – In the first round r where $\mathsf{GaveOutput}_i = \bot$ and $g_i^r = 2$, set $\mathsf{GaveOutput}_i = \top$ and output $(C_i, r_i) = (B_i^r, r)$. The output justifier is the justifier for (B_i^r, g_i^r).[a] If $\mathsf{P}_i \in \mathsf{C}_{\mathsf{EchoOutput}}$ it causal casts (B_i^r, g_i^r) with justifiers as a computed message.
   – In the first round r where $\mathsf{GaveOutput}_i = \bot$ and where some justified (B_j^ρ, g_j^ρ) propagated from $\mathsf{P}_j \neq \mathsf{P}_i$ with $g_j^\rho = 2$, set $\mathsf{GaveOutput}_i = \top$, and output $(C_i = B_j^\rho, \rho)$. The output justifier is the justifier for (B_j^ρ, g_j^ρ).

[a] Note that as the inputs had grade less than 2 this justifies the protocol not terminating earlier, forcing even corrupt parties to output a round number in which an honest party could have terminated.

Fig. 21. $\Pi_{\mathsf{SelectBlock}}$

Block Selection as subprotocol instead of Graded Block Selection[7]. So it still implements Justified Block Selection by the proof in [KN23]. We only need to argue that it additionally satisfies the Adjacent Output Round Agreement property.

Lemma 5 (Adjacent Output Round Agreement). *For all possible justified outputs of* $\Pi_{\mathsf{SelectBlock}}$ (C_i, r_i) *and* (C_j, r_j) *it holds that* $|r_i - r_j| \leq 1$.

Proof. Consider any output (C_i, r_i) justified by the justifier for $(B_i^{r_i} = C_i, g_i^{r_i} = 2)$, and any output (C_j, r_j) justified by the justifier $(B_j^{r_j} = C_j, g_j^{r_j} = 2)$. If $r_i = r_j$ we are done, so assume they are different and without loss of generality that $r_i \leq r_j$. Since the output in round r_i was justified by a grade 2, then by Graded Agreement all possible justified outputs from $\Pi_{\mathsf{WeakGradedSelectBlock}}$ in round r_i contain the same block. Which in turn by Strong Stability implies that all justified outputs in round $r_i + 1$ have grade 2, and thus that there are no justified input to $\Pi_{\mathsf{WeakGradedSelectBlock}}$ in round r_i+2. It follows that if $(\cdot, r)$ is a justified output of $\Pi_{\mathsf{SelectBlock}}$, then no honest party initiated $\Pi_{\mathsf{WeakGradedSelectBlock}}$ for any round $r' > r + 1$.

C.10 Justified Agreement on Core Set with Adjacent Output Round Agreement

We present a YOSO protocol for ACS Π_{ACS}, which again is almost identical to the protocol in [KN23], except that it is adapted to use YOSO committees and and adds a round number to its output. Since this is just the output from $\Pi_{\mathsf{SelectBlock}}$, Π_{ACS} inherits the Adjacent Output Round Agreement property. This just adds some auxiliary information to the output and has no effect on the validity of the proofs showing that it satisfies the remaining properties

[7] Note that the former is just a special case of the latter

of ACS. In conclusion Π_{ACS} satisfies Definition 14 which is identical to the ACS definition from [KN23], except that it includes the Adjacent Output Round Agreement property.:

Definition 14 (Justified ACS with Adjacent Output Round Agreement). *A protocol for M parties $\mathsf{P}_1, \ldots, \mathsf{P}_M$ with input and output justifiers J_{IN} and J_{OUT}. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at the time of input.*

Liveness: *If all honest parties start running the protocol with a J_{IN}-justified input then eventually all honest parties have a J_{OUT}-justified output.*

Validity: *For all possible J_{OUT}-justified outputs $(U, \cdot)$ and all honest P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that P_i had input B_i.*

Justified Blocks: *For all possible justified outputs $(U, \cdot)$ and all (potentially corrupt) P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that $J_{IN}(B_i) = \top$.*

Agreement: *For all possible justified outputs $(U_i, \cdot)$ and $(U_j, \cdot)$ it holds that $U_i = U_j$.*

Large Core: *For all possible justified outputs $(U, \cdot)$ it holds that $|S| \geq n - t$.*

Adjacent Output Round Agreement: *For all possible justified outputs (U_i, r_i) and (U_j, r_j) it holds that $|r_i - r_j| \leq 1$ and if $(\cdot, r)$ is a justified output then no honest party sent a message in any round $r' > r + 1$.*

Protocol Π_{ACS}

1. The input of P_i is B_i with $J_{\text{IN}}(B_i) = \top$.
2. If $\mathsf{P}_i \in \mathsf{C}_{\mathsf{ACSPropose}}$ is causal casts B_i. This message is justified by $J_{\text{IN}}(B_i) = \top$ and B_i having been reliably broadcast by P_i.
3. Party P_i collects at least $n-t$ justified B_j from parties $\mathsf{P}_j \in \mathsf{Collected}_i$ and lets $U_i = \{(\mathsf{P}_j, B_j)\}_{\mathsf{P}_j \in \mathsf{Collected}_i}$. This value is justified by each B_j being justified and $|U_i| \geq n - t$.
4. Run $\Pi_{\mathsf{SelectBlock}}$ where P_i inputs U_i. The input justifier of $\Pi_{\mathsf{SelectBlock}}$ is to check that U_i is justifiable as defined in the above step.
5. Party P_i gets output (C_i, r_i) from $\Pi_{\mathsf{SelectBlock}}$ and outputs (C_i, r_i). The output justifier is that (C_i, r_i) is a justified output from the above $\Pi_{\mathsf{SelectBlock}}$.

Fig. 22. Π_{ACS}

D Proofs for Total-Order Broadcast

In this section we will switch from simulation based security to game-based security and prove that the protocol Π_{TOB} defined in Fig. 13 satisfies the following definition.

Definition 15 (Game-based TOB). *We say that a protocol for M parties Π_{TOB} is a game-based secure TOB if for all PPT environments corrupting at most $T < (1 - \epsilon)M/3$ parties the following properties hold except with negligible probability for at random run $\text{Exec}_{\Pi_{\mathsf{TOB}}, \mathcal{E}}$ in the $(\mathcal{F}_{CF}, \mathcal{F}_{\text{ATOMIC-SEND}})$-hybrid model. Each party P holds a ledger L_{P}.*

Agreement *For any honest P and P', either $L_{\mathsf{P}} \sqsubseteq L_{\mathsf{P}'}$ or $L_{\mathsf{P}'} \sqsubset L_{\mathsf{P}}$.*

Validity *For each honest party P with ledger L_{P}. If (mid, m) is in a block in L_{P} and $P(\mathsf{mid}) = P_i$ and P_i is honest, then P_i sent (mid, m). Additionally, when looking at L_{P} as a sequence of blocks $B_1, \ldots, B_b$, the wait predicate W is satisfied for each prefix, i.e. $W^i(B_1 || \ldots || B_{i-1}, B_i)$ for each batch i.*

Liveness *Assuming new wait predicates W^i and messages satisfying them are continuously input to the protocol and that all honest parties get the same wait predicates in the same batches, then all messages input to the protocol are eventually added to L_{P} for all honest parties P.*

The UC and game based definitions are equivalent as long as there are no secret inputs to simulate, only correctness properties. Appendix D.1 elaborates.

Theorem 5. *The protocol Π_{TOB} described in Fig. 13 satisfies the game-based definition of Total-Order Broadcast in Definition 15. Assuming block size, $\alpha = \Omega(\lambda)$ it uses expected $O(M(\beta + \iota\lambda^2))$ bits of communication to order ι messages of combined size β in the coin flip hybrid model, and additionally needs setup for expected amortized $O(1)$ coin flips per batch to be computed to instantiate the coin flip.*

Proof. Agreement follows immediately from agreement of ACS and RB. Validity with respect to the individual messages in the block follows from the validity of RB, while the validity with respect to the wait predicate follows from all blocks input to ACS being justified by individually satisfying the wait predicate and the fact that a list of messages that satisfy the wait predicate will still satisfy it after permuting it or adding more message to it (cf. Section 5). Finally liveness follows from liveness of the subprotocols. Note that unless Π_{TOB} is continuously updated with new wait predicates that all parties agree on, the liveness property is an empty statement. $\Pi_{\mathsf{SelectBlock}}$ terminates in expected constant rounds, and since the protocol repurposes unused setups at most an expected constant number of new setups needs to be computed per batch. Each batch of the TOB consists of reliable broadcasts of the messages referenced in the block and one instance of Π_{ACS} to agree on which messages makes it into each batch and in which order. To produce new batch Π_{TOB} runs one instance of Π_{ACS} to agree on a set of blocks which references messages that were previously RBed. We first account for the cost RBing messages across all batches which for ι messages of combined size β is $O(M(\beta + \iota\lambda^2))$ when using Π_{RB}. In each batch a committee of expected λ parties RB a block which references within a constant of $\max(\alpha, W_\#)$ messages, which has communication complexity $O(\lambda M(\max(\alpha, W_\#) \log M + \lambda^2))$. Even if they all happen to reference the same same set of messages, this gives a per message cost of $O(\frac{M\lambda^3}{\max(\alpha, W_\#)})$, which since we assumed $\alpha = \Omega(\lambda)$ is $O(M\lambda^2)$ and thus dominated by the cost of RBing the message we accounted for above. Finally to run Π_{ACS} on the proposed blocks, a sequence of an expected constant number of committees of expected size λ need to send descriptions of subsets of the preceding committee through RB. This can again be done by each committee member sending a λ^2 bit list of public keys through the RB protocol which for each list has communication complexity $O(M\lambda^2)$. So, each invocation of Π_{ACS} contributes $O(M\lambda^3)$ bits of communication. Again since we assumed at least $\alpha = \Omega(\lambda)$ messages per epoch this $O(M\lambda^2)$ per message and dominated by the RB of the messages. In conclusion we get a

communication complexity of $O(M(\beta + \iota\lambda^2))$ to add ι messages of combined size β to the ledger, which is optimal if the average message size if $\Omega(\lambda^2)$.

D.1 Equivalence Between Game Based and UC TOB

The following theorem shows that game-based security implies simulation based security for TOB.

Theorem 6. *If Π_{TOB} is a game-based TOB then Π_{TOB} UC security implements $\mathcal{F}_{TOB}$ in the $(\mathcal{F}_{CF}, \mathcal{F}_{\text{ATOMIC-SEND}})$-hybrid model against $T < (1-\epsilon)M/3$ adaptive corruptions.*

Proof. We have to prove that there exists a PPT simulator $\mathcal{S}$ such that $\text{Exec}_{\Pi_{\mathsf{TOB}},\mathcal{E}} \approx \text{Exec}_{\mathcal{F}_{\text{TOB}},\mathcal{S},\mathcal{E}}$ for all PPT environments $\mathcal{E}$ doing at most $T < (1-\epsilon)M/3$ adaptive corruptions. The simulator $\mathcal{S}$ proceeds as follows. Note that whenever an input is given to $\mathcal{F}_{\text{TOB}}$ information is leaked which allows $\mathcal{S}$ to compute this input. The simulator will run Π_{TOB} on these inputs including copies of $\mathcal{F}_{\text{CF}}$ and $\mathcal{F}_{\text{ATOMIC-SEND}}$. It lets the environment $\mathcal{E}$ interact with $\mathcal{F}_{\text{CF}}$ and $\mathcal{F}_{\text{ATOMIC-SEND}}$ as in $\text{Exec}_{\Pi_{\mathsf{TOB}},\mathcal{E}}$. Whenever the copy of Π_{TOB} run by $\mathcal{S}$ produces an output then $\mathcal{S}$ uses its influence over $\mathcal{F}_{\text{TOB}}$ to make the copy of $\mathcal{F}_{\text{TOB}}$ in $\text{Exec}_{\mathcal{F}_{\text{TOB}},\mathcal{S},\mathcal{E}}$ produce the same outputs to the same parties as Π_{TOB}. To be able to do this it is clearly enough that Π_{TOB} is a game-based TOB, as this ensures its outputs are always possible outputs of $\mathcal{F}_{\text{TOB}}$. □

When we work with game-based security definitions for sub-protocols, in all cases the properties are tacitly meant to hold hold except with negligible probability for all PPT environments corrupting at most $T < (1-\epsilon)M/3$ parties and for a random run $\text{Exec}_{\Pi_{\mathsf{TOB}},\mathcal{E}}$ in the $(\mathcal{F}_{\text{CF}}, \mathcal{F}_{\text{ATOMIC-SEND}})$-hybrid model.

E Two-level ciphertexts.

In this section, we show how parties in a committee can generate random ciphertexts of form $E_{N,w_{i+1}}(E_{N,w_i}(0))$ for $i = 1 \ldots \lambda - 1$ and prove they are correctly formed. These can then be combined to get a random two-level encryption of 0, and such a ciphertext can in turn be used to randomize encryptions in the multiparty version of the PIR protocol.

For this purpose, we will need that a party P, can publish a "two-level" encryption of 0: $c = E_{N,w_{i+1}}(E_{N,w_i}(0))$ and give a non-interactive zero-knowledge proof that the ciphertext was correctly formed. We will denote such a proof for the above relation by $\mathsf{NIZK}((r,s) : c = E_{N,w_{i+1}}(E_{N,w_i}(0;r);s))$. Below, we present and analyse a Σ-protocol for this relation which can then be made non-interactive in the random oracle model, as usual. The protocol is novel, and is an application of techniques known from so-called double discrete log proofs.

Random Two-level Encryptions The protocol in Fig. 23 produces a random two-level encryption of 0. Several instances can be run in parallel to create any desired number of outputs.

Protocol RandTwoLevel

1. On input i, where $0 \leq i < S$, Each member P_u of the current committee computes $d_u = E_{N,w_{i+1}}(E_{N,w_i}(0; r_u); s_u)$ and sends d_u, π_u to $\mathcal{F}_{\text{TOB}}$, where
$$\pi_u = \mathsf{NIZK}((r_u, s_u) : d_u = E_{N,w_{i+1}}(E_{N,w_i}(0; r_u); s_u)) \ .$$
2. Once $h \geq n - t$ valid contributions $(d_1, \pi_1), \ldots, (d_h, \pi_h)$ are delivered from $\mathcal{F}_{\text{TOB}}$, output $d = \mathsf{Multiply}(d_1, \ldots, d_h)$

Fig. 23. The protocol for generating random two-level encryptions of 0

For correctness of RandTwoLevel, note that the plaintext inside the output ciphertext d is the product of the plaintexts inside the d_u's which are themselves ciphertexts. Hence, d is an encryption of

$$\prod_{u=0}^{t} E_{N,w_i}(0; s_u) \bmod N^{i+1} = E_{N,w_i}(0; \prod_{u=0}^{t} s_u \bmod N).$$

So, d is itself a two-level encryption of 0, where at least one contribution to the randomness comes from an honest player since there are $t + 1$ contributions. We can therefore think of the protocol as implementing a functionality that produces random two-level encryptions of 0.

The protocol in Fig. 24 allows a party, here denoted by P, to publish a two-level encryption of 0: $E_{N,w_{i+1}}(E_{N,w_i}(0))$ and convince a verifier V in (honest verifier) zero-knowledge that is correctly formed. This protocol can then be turned into a non-interactive proof using standard tools, to form the proof needed in the RandTwoLevel protocol

Protocol TwoLevel.

1. P sends the ciphertext $c = E_{N,w_{i+1}}(E_{N,w_i}(0; r); s)$ to V, and then the proof below is executed k times in parallel.
2. (a) P computes $E_{N,w_i}(0; r')$ for random r' and sends to V
$$a = c^{E_{N,w_i}(0;r')} s'^{N^{i+1}} \bmod N^{i+2} = E_{N,w_{i+1}}(E_{N,w_i}(0; rr' \bmod N); ss' \bmod N)$$
   (b) V sends a random bit e to P.
   (c) If $e = 0$, P sends $z_1 = r'$ and $z_2 = s'$ to V. If $e = 1$, P sends $z_1 = rr' \bmod N$ and $z_2 = ss' \bmod N$.
   (d) If $e = 0$, V checks that $a = c^{E_{N,w_i}(0;z_1)} z_2^{N^{i+1}} \bmod N^{i+2}$.
   If $e = 1$, V checks that $a = E_{N,w_{i+1}}(E_{N,w_i}(0; z_1); z_2)$.

Fig. 24. The Σ-protocol for two-level ciphertexts

Theorem 7. *For any $i = 0, \ldots, S-1$, the* TwoLevel *protocol is a Σ-protocol for the relation*

$$\{((N, w_i, w_{i+1}, c), (r, s)) |\ c = E_{N,w_{i+1}}(E_{N,w_i}(0; r); s)\},$$

i.e., P knows r, s such that $c = E_{N,w_{i+1}}(E_{N,w_i}(0; r); s)$ and the protocol is complete and perfect honest verifier zero-knowledge.

Proof. Completeness is clear from inspection of the protocol. For special soundness (which is well-known to imply standard knowledge soundness), we assume we are given two accepting conversations (a, e, z_1, z_2) and (a, e', z_1', z_2') with $e \neq e'$, and must show that we can efficiently find valid values for r, s. Assume without loss of generality that $e = 0$. Then we have

$$c^{E_{N,w_i}(0;z_1)} z_2^{N^{i+1}} \bmod N^{i+2} = a = E_{N,w_{i+1}}(E_{N,w_i}(0; z_1'); z_2')$$

which implies

$$c^{E_{N,w_i}(0;z_1)} = E_{N,w_{i+1}}(E_{N,w_i}(0; z_1'); z_2' z_2^{-1} \bmod N)$$

One can now compute $E_{N,w_i}(0; z_1)^{-1} \bmod N^{i+1}$, and observe that for some integer w we have $E_{N,w_i}(0; z_1)^{-1} \cdot E_{N,w_i}(0; z_1) = 1 + wN^{i+1}$. Now, by raising both side of the above equation to $E_{N,w_i}(0; z_1)^{-1}$, we get

$$c = E_{N,w_{i+1}}(E_{N,w_i}(0; z_1' z_1^{-1} \bmod N); z_2' z_2^{-1} c^{-w} \bmod N) \ .$$

Thus we see that we can compute valid values for r, s given the two conversations.

Finally, for honest verifier zero-knowledge, we show a simulator which first chooses random bit e and random $z_1, z_2 \in \mathbb{Z}_N^*$. Then, if $e = 0$ it sets $a = c^{E_{N,w_i}(0;z_1)} z_2^{N^{i+1}} \bmod N^{i+2}$. If $e = 1$ it sets $a = E_{N,w_{i+1}}(E_{N,w_i}(0; z_1); z_2)$, and finally, it outputs (a, e, z_1, z_2). It is clear that e, z_1 and z_2 have the right distribution, and a is set to be the only correct value, given e, z_1 and z_2. Hence the simulation is perfect. □

F Linear Integer Secret Sharing

The material on linear integer secret sharing in this section is taken from [DT06], while the later section on non-interactive VSS is a contribution of this paper.

We will need to secret share the secret decryption exponent which of course "lives in the exponent". So, the natural choice would be a sharing scheme that uses arithmetic modulo the order of the group. However, all parties need to be able to run the scheme and the group order is not public knowledge, so we resort to secret-sharing over the integers instead. Earlier work has used integer variants of Shamir's scheme, but this leads to technical difficulties due to the fact that the Lagrange coefficients needed for interpolation are not integers. In earlier work, this implies that the size of the shares grow each time the secret key is re-shared. We will instead use a linear integer sharing scheme (LISS), which allows us to avoid this problem and simplify the protocols, at the cost of a larger share size in the beginning.

A LISS is defined by a *sharing matrix* M with integer entries, c columns and ℓ rows (one can think of as replacing the Van der Monde matrix from Shamir's scheme). One can share a secret number $s \in [0, 2^b]$ for a publicly known upper bound b among n players. To do this, choose a column vector $\boldsymbol{v}_s$ with s as the first entry and sufficiently large random numbers in the other entries (we make this precise in a moment). Shares are computed as the product

$M \cdot \boldsymbol{v}_s$. This corresponds to evaluating the polynomial in a number of points in Shamir's scheme.

Each row of M is labelled with an index in $[1, n]$, we say that each row is owned by a player. Our notation for this is that row i is owned by player number $u(i)$. Each entry in $M \cdot \boldsymbol{v}_s$ corresponds to a row, and the entry is handed to the player who owns that row. We will refer to an entry in $M \cdot \boldsymbol{v}_s$ as a *share*, but note that each player may receive several shares.

For each player set A we let M_A denote the matrix we obtain by selecting from M only the rows owned by players in A. If A is qualified to reconstruct the secret, there exists a *reconstruction vector* $\boldsymbol{r}_A$ with the property that

$$\boldsymbol{r}_A \cdot (M_A \cdot \boldsymbol{v}_s) = s,$$

this corresponds to the interpolation in Shamir's scheme.

If A is not qualified to reconstruct, there exists a *sweeping vector* $\boldsymbol{w}_A$, which has 1 as it's first entry, and further has the property that $M_A \cdot \boldsymbol{w}_A$ is the all-0 vector. It can be shown that the existence of $\boldsymbol{w}_A$ implies statistical privacy of the scheme. Namely, we define w_{max} to be the maximal numeric value of any entry occurring in any $\boldsymbol{w}_A$. Then, a *valid sharing vector* for $s \in [0, 2^b]$ is a vector $\boldsymbol{v}_s$ with s in the first entry, and with the other entries chosen uniformly from $[0, 2^{b+\log_2(w_{max})+k}]$, where k is the security parameter.

Lemma 6. *For any two secrets $s, s' \in [0 \dots 2^b]$, the distributions of shares seen by A from sharing s or s', with valid sharing vectors, are statistically indistinguishable.*

Proof. (Sketch) if s was shared using $\boldsymbol{v}_s$, one could instead have shared s' using $\boldsymbol{v}_s + (s' - s)\boldsymbol{w}_A$, and the players in A would receive exactly the same shares. But the numbers in $\boldsymbol{v}_s$ are a factor 2^k larger than those in $(s' - s)\boldsymbol{w}_A$, so $\boldsymbol{v}_s + (s' - s)\boldsymbol{w}_A$ is statistically indistinguishable from a valid sharing vector for s', so the lemma follows.

It has been shown [DT06] that LISS schemes with polynomial share size exist for the threshold case, where any majority of the players are qualified to reconstruct, this follows from the fact that polynomial size monotone formulae exist for computing the majority function. We let M_{th} denote such a scheme for n players. It is also straightforward to see that simple additive secret sharing can be realized in this formalism, using a matrix M_{add} with one $n + 1$ columns and n rows. namely, M_{add} is the identity matrix, except that the first row has -1 in the last n entries. Notably, for these schemes, all entries in all sweeping vectors are 1 or -1, so the numbers in a valid sharing vector just need to be k bits longer than the secret.

In this paper, we will need to share a secret among the members of two committees, that we will call the *additive committee* and the *threshold committee*. This is needed for technical reasons, to obtain adaptive security. The idea is to share the secret additively among the members of the additive committee, and then each additive share is shared among the members of the threshold committee. It is not hard to see that this can be phrased as one LISS scheme using a matrix M that we can build from M_{add} and M_{th}. We will not do the straightforward (but very tedious) details of this.

Note that, in the scheme defined by M the sets qualified to reconstruct the secret will be, either all members of the additive committee, or a majority of the threshold committee.

Verifiable Secret Sharing We require a verifiable LISS scheme that can be used to do distributed exponentiation in the groups $\mathbb{Z}^*_{N^{s+1}}$, for $s = 1, \ldots, S$. We will use a Pedersen-style construction for this, where the idea is that the secret and the sharing vector are committed to using the integer commitment scheme $\mathsf{Com}_{\mathsf{ck}}(\cdot;\cdot)$ we described in the preliminaries. We then define the algorithm $\mathsf{VSSshare}$, in figure 25.

Algorithm VSSshare.

1. To share a secret d, choose a valid sharing vector $\boldsymbol{v}_d$ and compute $\mathsf{sh}(d, \boldsymbol{v}_d) = M \cdot \boldsymbol{v}_d$.
2. Choose a vector $\boldsymbol{r}$ containing randomness values for the commitment scheme and compute commitments to each entry $\boldsymbol{v}_d[j]$ as $\beta_j = \mathsf{Com}_{\mathsf{ck}}(\boldsymbol{v}_d[j]; \boldsymbol{r}[j])$ as well as $\mathsf{ra}(\boldsymbol{r}) \coloneqq M \cdot \boldsymbol{r}$.
3. Output
$$\mathsf{VSS}(d, \boldsymbol{v}_d, \boldsymbol{r}) = (\mathsf{sh}(d, \boldsymbol{v}_d), \mathsf{ra}(\boldsymbol{r}), \beta_1, \ldots \beta_c).$$

Fig. 25. The VSSshare algorithm.

Note that $\beta_1 = \mathsf{Com}_{\mathsf{ck}}(d; \boldsymbol{r}[1])$ and so serves as a commitment to the secret.

Assume a dealer distributes the shares in $\mathsf{sh}(d, \boldsymbol{v}_d)$ to the players who own them, as well as the corresponding values in $\mathsf{ra}(\boldsymbol{r})$, and makes the β_j's public. Then the players can verify their shares. Namely, let $\boldsymbol{m}_i$ be the ith row of M, then $\mathsf{sh}(d, \boldsymbol{v}_d)[i] = \boldsymbol{m}_i \cdot \boldsymbol{v}_d$, and $\mathsf{ra}(\boldsymbol{r})[i] = \boldsymbol{m}_i \cdot \boldsymbol{r}$. So, using the β_j's and $\boldsymbol{m}_i$, the holder of the i'th share can compute a commitment α_i to his share and verify that she has been given information to correctly open it. The check is done by verifying that:

$$\alpha_i := \prod_{j=1}^{c} \beta_j^{\boldsymbol{m}_i[j]} = \mathsf{Com}_{\mathsf{ck}}(\mathsf{sh}(d, \boldsymbol{v}_d)[i]; \mathsf{ra}(\boldsymbol{r})[i]) \tag{2}$$

This equation should hold by the homomorphic property of the commitments. Indeed, we have:

$$\begin{aligned}
\alpha_i &= \prod_{j=1}^{c} \beta_j^{\boldsymbol{m}_i[j]} && (3)\\
&= \prod_{j=1}^{c} (\mathsf{Com}_{\mathsf{ck}}(\boldsymbol{v}_d[j]; \boldsymbol{r}[j]))^{\boldsymbol{m}_i[j]} && (4)\\
&= \mathsf{Com}_{\mathsf{ck}}(\boldsymbol{m}_i \cdot \boldsymbol{v}_d; \boldsymbol{m}_i \cdot \boldsymbol{r}) && (5)\\
&= \mathsf{Com}_{\mathsf{ck}}(\mathsf{sh}(d, \boldsymbol{v}_d)[i]; \mathsf{ra}(\boldsymbol{r}[i])). && (6)
\end{aligned}$$

We explain below how this check ensures that the secret is well-defined and can be reconstructed.

Non-interactive VSS In the usual form of a VSS, players would receive shares and complain if they fail verification. However, we will need a non-interactive VSS, i.e., a possibly corrupt dealer can broadcast a single message to a set of n players and as a result they obtain valid shares of the secret the dealer had in mind, or they all conclude that the dealer is corrupt. The idea for this is straightforward: the dealer will compute $\mathsf{VSS}(d, \boldsymbol{v}_d)$, encrypt the individual shares for each player, and attach non-interactive zero-knowledge proofs that the correct shares are encrypted. We assume that the public keys of the share holders $P_1, \ldots, P_n$ are $\mathsf{gpk}_1, \ldots, \mathsf{gpk}_n$, these are keys for the El Gamal-style encryption we defined before.

Then, given $\alpha = \mathsf{Com}_{\mathsf{ck}}(z; v)$ for some z, v and a ciphertext $c = E_{\mathsf{gpk}_u}((z, v); r)$, we require a non-interactive zero-knowledge proof of knowledge that the ciphertext contains the integers z, v used in forming α, and also that z is in a bounded interval $[0, 2^a]$. We denote such a proof by

$$\mathsf{NIZK}(z, r, v : \alpha = \mathsf{Com}_{\mathsf{ck}}(z; v),\ c = E_{\mathsf{gpk}_u}((z, v); r),\ z \in [0..2^a])$$

In the following we will use this proof in a case where z is a share of some secret. The secret will be verified to be at most 2^b, so it follows that the entries in the sharing vector should be chosen from a bounded size interval as explained above, and given the sharing matrix M this implies an upper bound on the size of any share, we denote this bound by $2^{sh(b)}$, and will use it as the bound 2^a in the proof.

We also require a non-interactive zero-knowledge proof of knowledge that the prover knows how to open a set of commitments $\beta_1, \ldots, \beta_c$, and that the integer committed to in β_1 is in the interval $[0, 2^b]$, recall that the secret is committed to via β_1. We denote such a proof by

$$\mathsf{NIZK}(\{a_j, b_j\}_{j=1}^c : \{\beta_j = \mathsf{Com}_{\mathsf{ck}}(a_j; b_j)\}_{j=1}^c,\ a_1 \in [0, 2^b])$$

We will assume that the proof systems are statistical zero-knowledge, on-line extractable and unconditionally simulation sound, as explained in Section 2. A dealer in our VSS will proceed using the NonIntVSSshare algorithm, in Figure 26.

Algorithm NonIntVSSshare

1. To share a value $d \in [0, 2^b]$, first compute $\mathsf{VSS}(d, \boldsymbol{v}_d, \boldsymbol{r}) = (\mathsf{sh}(d, \boldsymbol{v}_d), \mathsf{ra}(\boldsymbol{r}), \beta_1, \ldots \beta_c)$ as defined above.
2. Compute
$$\pi_0 = \mathsf{NIZK}(\{\boldsymbol{v}_d[j], \boldsymbol{r}[j]\}_{j=1}^c : \{\beta_j = \mathsf{Com}_{\mathsf{ck}}(\boldsymbol{v}_d[j]; \boldsymbol{r}_j)\}_{j=1}^c,\ \boldsymbol{v}_d[1] \in [0, 2^b]).$$
3. For each $i = 1 \ldots \ell$, let $s_i = \mathsf{sh}(d, \boldsymbol{v}_d)[i]$ and $v_i = \mathsf{ra}(\boldsymbol{r})[i]$. Use equation 2 to compute $\alpha_i = \mathsf{Com}_{\mathsf{ck}}(s_i; v_i)$. Compute a ciphertext $c_i = E_{\mathsf{gpk}_{u(i)}}((s_i, v_i); r_i)$, where $\mathsf{gpk}_{u(i)}$ is the public key of the receiver of the i'th share.
4. For $i = 1 \ldots \ell$, compute
$$\pi_i = \mathsf{NIZK}(s_i, r_i, v_i : \alpha_i = \mathsf{Com}_{\mathsf{ck}}(s_i; v_i),\ c_i = E_{\mathsf{gpk}_{u(i)}}((s_i, v_i); r_i),\ s_i \in [0..2^{sh(b)}]).$$
5. Output $\mathsf{NonIntVSSshare}(2^b, d, \boldsymbol{v}_d, \boldsymbol{r}, \mathsf{gpk}_1, \ldots, \mathsf{gpk}_n) := (\beta_1, \ldots, \beta_c, c_1, \ldots, c_\ell, \pi_0, \pi_1, \ldots, \pi_\ell)$.

Fig. 26. The NonIntVSSshare algorithm.

Anyone can then use the algorithm VSSverify, in Figure 27, to check what the dealer sent.

Algorithm $\mathsf{VSSverify}(2^b)$

1. Given $\mathsf{NonIntVSSshare}(2^b, d, \boldsymbol{v}_d, \boldsymbol{r}, \mathsf{gpk}_1, \ldots, \mathsf{gpk}_n) = (\beta_1, \ldots \beta_c, c_1, \ldots, c_\ell, \pi_0, \pi_1, \ldots, \pi_\ell)$, Verify π_0 against public values $\beta_1, \ldots, \beta_c$. For $i = 1 \ldots \ell$ compute α_i using equation (2), verify the proof π_i against public values α_i and c_i. Abort if any proof fails.

Fig. 27. The VSSverify algorithm.

If the dealer's message verifies, a receiver P_u of shares will decrypt all ciphertexts c_i for which $u = u(i)$ and store the resulting set of shares. We say that the honest players *hold correct shares*, if there exists $d, \boldsymbol{v}_d$ such that for all s_i held by $P_{u(i)}$ is honest, we have $s_i = \mathsf{sh}(d, \boldsymbol{v}_d)[i]$

Lemma 7. *The event that a corrupt dealer's message verifies but honest players do not hold correct shares, happens with negligible probability.*

Proof. Suppose we are given an adversary that can make the event specified in the lemma happen with non-negligible probability. We show that such an adversary can be used to break the binding property of the commitment scheme. We take a public commitment key ck as input and run the adversary with this key. Note that the zero-knowledge proofs given are straight-line extractable. This concretely means that if the dealer's message verifies, then (except with negligible probability) from the dealer's oracle calls and the proof π_0, one can extract opening information for all the β_j's, to get a sharing vector $\boldsymbol{v}_d$. In particular we can extract d from β_1. Then, from equation (2), one can compute opening information for all the α_i's and by construction of equation(2) this will show how to open the α_i's to reveal a set of valid shares $\{s_i\}$ of d, where $s_i = \mathsf{sh}(d, \boldsymbol{v}_d)[i]$. On the other hand, from the proof π_i one can extract the content of c_i and this will also reveal a way to open α_i, to get a value s_i'. Thus, if the adversary's attack is successful, it must be that for some i where $P_{u(i)}$ is honest, $s_i \neq s_i'$. We can then output the two ways to open α_i and break binding.

Note that the reduction from the above proof can be used in context of our global protocol because the commitment key is given as set-up, and is chosen independently from everything else.

We note that we will not require an explicit reconstruction protocol, instead the shares held by the players will be used for decryption as detailed in the main protocol. Basically, all (honest) players in the additive committee will contribute to the decryption of a given ciphertext. Since contributions from the entire additive committee would be required to decrypt, we have the threshold committee fill in for the those additive players that do not contribute.

As for privacy, the intuition is clear: an unqualified set of shares reveals essentially nothing about the secret, and the commitments and zero-knowledge proofs leak (statistically) no

additional information. However, we will need to show adaptive security of the global protocol, so we need to be careful. To this end, we will use in our security proof given elsewhere the so-called single inconsistent player (SIP) technique. Here, the idea is that the simulator is set up such that it knows a complete set of shares for players in the additive and threshold committees, however, these shares determine some dummy value. The simulator selects at random a player from the additive committee to be the SIP. When simulating decryption, it can fake the contribution from the SIP to make the output plaintext be correct. Therefore, as long as the SIP is not corrupted and its contribution is delivered on time, the simulation will be statistically indistinguishable. This happens with probability at least $1/n$.

G Σ-protocols and Non-interactive Zero-Knowledge Proofs

In this section we sketch the (well-known) techniques one can use to get Σ-protocols for the relevant relations. Once we have these, we can get NIZK's in the random oracle model, using the Fischlin transform [Fis05].

For the case of two-level Paillier ciphertexts, the required protocol was already described and analysed in.

Otherwise, we use two main types of proofs in our protocols, namely those that are used in the Role assignment and MPC protocols, and those that are used in the Decryption and Reshare protocols.

The first type of proofs work only on Paillier ciphertexts, and the protocols we need for this can be found in [DJN10], they can be used here with no essential change.

The second type of proofs work with the personal keys of players gpk_i, and the integer commitments we use and involve showing that, for a commitment $\mathsf{Com}_{\mathsf{ck}}(z;r)$, z, r have been encrypted under gpk_i and that this z and r are in a certain interval.

Recall that a commitment to integer x is of form $\alpha^x \beta^r \bmod N'$, where r is randomly chosen in a large enough interval (we do not need the low-level details of the scheme for this discussion). We will later use $\mathsf{Com}(z)$ as shorthand for a commitments to z. Note that commitments are linearly homomorphic, we have $\mathsf{Com}(z) \cdot \mathsf{Com}(z')^f = \mathsf{Com}(z + fz') \bmod N'$.

Consider now the proof we require in the Reshare protocol where we are given $\mathsf{Com}_{\mathsf{ck}}(z;r)$ for some z and ciphertexts $c_z = E_{\mathsf{gpk}_u}(z; v_z), c_r = E_{\mathsf{gpk}_u}(r; v_r)$, we require a non-interactive zero-knowledge proof of knowledge of x, r, v_z and v_r such that commitment and ciphertext are of the claimed form.

To build a Σ-protocol for this setting, the prover does a number of different Σ-protocols in parallel. We first explain how we establish connection from the commitment to the ciphertext containing z. First, known techniques from [DF02] are used to show that the prover knows z. The prover also writes z as $z = \sum_{j=0}^{a} 2^{\gamma j} z_j$, where $z_1, \ldots, z_a$ are of size γ bits. Recall that $E_{\mathsf{gpk}_u}(z; r_z)$ is actually a tuple $\{E_{\mathsf{gpk}_u}(z_j; r_j)\}$ of encryptions of the z_j's where the bit size γ is chosen small enough that decrypting each individual z_i is feasible. The prover makes commitments $\mathsf{Com}(z_j)$, proves he knows the z_j and that they are in the interval $[0, 2^\gamma]$, again using techniques from [DF02]. The verifier checks that $\mathsf{Com}(z) = \prod_j \mathsf{Com}(z_j)^{2^{\gamma j}}$. Note that this establishes the equation $z = \sum_{j=0}^{a} 2^{\gamma j} z_j$, and therefore also implies that z is in the

required range. Finally the prover shows for each j that the commitment $\mathsf{Com}(z_j)$ contains the same integer as the ciphertext $E_{\mathsf{gpk}_u}(z_j; r_j)$. Since the z_j and the randomness use are all in the exponent, this is easily done using known techniques for proving equality of discrete logs.

For the ciphertext containing r, the prover makes an additional commitment $\mathsf{Com}(r)$ and shows that this was the value occurring in $\mathsf{Com}_{\mathsf{ck}}(z; r)$. This is easy using known techniques for equality of discrete logs. Then proceed exactly as above using $\mathsf{Com}(r)$ instead of $\mathsf{Com}(z)$ and c_r instead of c_z.

There is, however, one technicality we need to address: we want, of course, that the protocol has negligible soundness error, and we want to avoid having to use binary challenges and repeat protocols many times. For this to work out, it needs to be the case that all group elements involved in the statement to prove are in a group with only exponentially large prime factors in the order. This is potentially an issue as these elements may be adversarially generated. For α, β we are fine, as they are generated honestly by the set-up. For the ciphertext $E_{\mathsf{gpk}_u}(z_j; r_j)$, we would be fine if we knew that its constituents were in the subgroup of squares in $\mathbb{Z}_N^*$. But we cannot verify membership in this subgroup efficiently. To solve this, we simply square the elements of the ciphertext and do the proof on the result. More concretely, we have $E_{\mathsf{gpk}_u}(z_j; r_j) = (g_u^{r_j}, g_u^{z_j} h_u^{r_j} \bmod N)$. We now define

$$\begin{aligned} E_{\mathsf{gpk}_u}(z_j; r_j)^2 &= ((g_u^{r_j})^2 \bmod N, (g_u^{z_j} h_u^{r_j})^2 \bmod N) \\ &= ((g_u^2 \bmod N)^{r_j} \bmod N, ((g_u^2 \bmod N)^{z_j} (h_u^2 \bmod N)^{r_j} \bmod N) \end{aligned} \tag{7}$$

We can see that $E_{\mathsf{gpk}_u}(z_j; r_j)^2$ only contains elements in the group of squares, moreover, if the ciphertext is honestly generated, it is an encryption of the same plaintext z_j under the public key $(g_u^2 \bmod N, h_u^2 \bmod N)$, and finally it can be decrypted using the original secret key. Therefore, if in the proof we replace $E_{\mathsf{gpk}_u}(z_j; r_j)$ by $E_{\mathsf{gpk}_u}(z_j; r_j)^2$ and the public key by $(g_u^2 \bmod N, h_u^2 \bmod N)$, we are guaranteed negligible soundness error and everything else works the same way.

Finally, for the proofs used in the decryption protocol, we observe that these are again simple applications of equality of discrete log techniques, as all the secret parameters are in the exponent. This is clear for the case of public decryption.

We need to address the same technicality as before, that the involved ciphertexts may be adversarially generated and so might not be in the subgroup they are supposed to. We solve this by the same trick of squaring before we do the proof.

H Supplementary UC Formalization

H.1 Eventual Liveness

When modelling asynchronous security we will as usual have to talk about an event *eventually* happening, for instance saying that if a message is sent then it is eventually delivered. We now discuss how we model this. When specifying an ideal functionality we will as usual let the adversary specify when certain events E happen. We sometimes say that under certain

preconditions C the adversary must *eventually* make E happen. This means that whenever C becomes true it holds at some future point in time that either C stopped being true or E became true. As an example we might say that if m was sent from an honest party which is still honest, then eventually m will be delivered, where C is "m was sent by an honest party which is still honest and" and E is "m was delivered by the adversary". We call this an eventual event. We say that a protocol π using ideal functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$ and implementing $\mathcal{F}$ is live if it holds that when all eventual events on $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$ happened, then π made all eventual events of $\mathcal{F}$ happen. This way we do not need to define what it means for an event to eventually happen, rather we just require that protocols are "eventuality-preserving". As an example, if $\mathcal{F}_1$ is a functionality for sending messages on point-to-point channels and $\mathcal{F}$ the ideal functionality for broadcast then eventuality-preserving liveness could be of the form "if all messages sent on the point-to-point channels by honest parties in π have been delivered then all messages broadcast by honest parties via π have been delivered". This is a crude model but good enough for our study.

H.2 Ideal Functionality for Atomic Send

The ideal functionality for atomic send is given in Fig. 28.

Functionality $\mathcal{F}_{\text{ATOMIC-SEND}}$

Init: Let $\mathsf{Accepted} = \emptyset$.

Broadcast: On input $(\text{ATOMIC-SEND}, (\mathsf{mid}_1, m_1, P_1), \ldots, (\mathsf{mid}_\ell, m_\ell, P_1))$, where $P(\mathsf{mid}_i) = P$ for $i = 1, \ldots, \ell$, add each $(\mathsf{mid}_i, m_i, P_i)$ to $\mathsf{Accepted}$ and leak $(\text{ATOMIC-SEND}, (\mathsf{mid}_1, m_1), \ldots, (\mathsf{mid}_\ell, m_\ell))$ to the adversary. We assume that honest parties use each mid at most once.

Deliver: On input $(\text{DELIVER}, (\mathsf{mid}, m, R))$, where $(\mathsf{mid}, m, R) \in \mathsf{Accepted}$ or $P(\mathsf{mid})$ is currently corrupted, remove (mid, m, R) from $\mathsf{Accepted}$ and output (mid, m) to R.

Eventual Liveness: If $(\mathsf{mid}, m, R) \in \mathsf{Accepted}$ the adversary must eventually input $(\text{DELIVER}, (\mathsf{mid}, m, R))$.

Fig. 28. Atomic Send

H.3 Discussion of $\mathcal{F}_{\text{TOB}}$

For each batch the parties input a wait predicate indicating which messages should be collected. We assume that all honest parties agree on the wait predicate W for each block. That means that we only prove an implementation secure under this condition and that, on the other hand, when we use $\mathcal{F}_{\text{TOB}}$ in a protocol π then π must guarantee that all honest parties input the same W to $\mathcal{F}_{\text{TOB}}$. We cannot guarantee that all messages attempted to be sent are delivered in a given round, as the network is asynchronous. We therefore let the adversary choose which messages are delivered. This happens in **Next Batch**, where it picks the next block. However, the adversary is only allowed to pick a block valid by W. Note that this means that if the messages input by the honest parties do not satisfy W then $\mathcal{F}_{\text{TOB}}$ might deadlock. This is a feature. It is the obligation of the protocol using $\mathcal{F}_{\text{TOB}}$ to

pick W and send messages m such that W gets satisfied in each round. On the other hand this allows an implementation of $\mathcal{F}_{\text{TOB}}$ to wait until it saw messages satisfying W. Note that $\mathcal{F}_{\text{TOB}}$ is guaranteed to produce the next block once all honest parties requested it and their messages satisfy W. It might, however, produce the block *before* all honest parties ordered the block. We could not possibly wait for all honest parties to have ordered the block in an asynchronous network. Notice, however, that at least one honest party must have ordered the next block before it can be produced as W^{b+1} needs to be defined. The wait predicate W^{b+1} can therefore be used to control that the block is not produced too early—it might for instance say that the next block is valid only if there are messages from enough parties that at least one must be honest. Notice, finally that even though we cannot ensure that a message sent in round b will make it into block $b+1$ we *do* require that any message sent will eventually make it into *some* block. Finally, we require that all blocks are eventually delivered to all honest parties. This all in all means that all messages broadcast by honest parties are eventually delivered to all honest parties.

Looking forward, the implementation for $\mathcal{F}_{\text{TOB}}$ will use small committees and YOSO role assignment. It might seem puzzling that this is not reflected in $\mathcal{F}_{\text{TOB}}$. However, the committees are an implementation detail, not a part of the specification of total-order broadcast.

H.4 Threshold Coin-Flip

The ideal functionality for coin-flip is given in Fig. 29. It can trivially be implemented given $\mathcal{F}_{\text{RA+MPC+CF}}$. We introduce it as a separate ideal functionality to not use $\mathcal{F}_{\text{RA+MPC+CF}}$ in its full glory when implementing $\mathcal{F}_{\text{TOB}}$ from coin-flip.

Functionality $\mathcal{F}_{\text{CF}}$ for coin flip.

Init: For each party P let $\ell_P = 0$ be the number of coins delivered at P and let $b_P = 0$ be the number of coins ordered by P. Let $b = 0$ be the number of coins flipped so far. Let L be the empty ledger.

Order next coin: On input (NEXT-COIN) from honest P leak (NEXT-COIN, P) to the adversary, let $b_P = b_P + 1$, flip uniformly random $c_{b_P} \in \{0,1\}^\kappa$ if c_{b_P} was not already flipped, and give c_{b_P} to the adversary.

Coin Index: On input (COIN-INDEX) from honest P output b_P to P.

Next coin: On input (NEXT-COIN) from the adversary where $b_P > b$ for some honest P, let $b \leftarrow b + 1$, and let $L = L \| c_b$.

Deliver: On input (DELIVER, P) from the adversary where $\ell_P < |L|$ update $\ell_P = \ell_P + 1$ and output $(\ell_P, L[\ell_P])$ to P.

Eventual Liveness: If $b_P > b$ for all honest P the adversary eventually calls **Next coin** again. Furthermore, if $\ell_P < |L|$ then eventually the adversary inputs (DELIVER, P) again.

Fig. 29. Threshold Coinflip

I Basic Protocol for Secure Computation

This section contains some basic protocols for secure computation that we will need for the implementation of $\mathcal{F}_{\text{RA+MPC}}$. Many of these are standard, but we incorporate some new ideas, as accounted for in the text below.

Secure Multiplication. For secure multiplication, we use the standard idea of producing multiplication triples and using one triple later for each multiplication. The protocol for making triples is in Figure 30. It lets all parties compute the output triple locally from data on the ledger. Once a triple is produced it is placed on an ordered list, and when we say in the following that we use the next available triple, we mean that all parties take the next unused triple on the list.

Protocol Triple.
This protocol is parametrized by s, and will produce ciphertexts in $\mathbb{Z}_{N^{s+1}}$.

1. Each member P_u of the committee assigned to do this protocol instance chooses random a_u, r_u, computes $c_u = E_{N,w_s}(m_u; r_u)$ and sends to $\mathcal{F}_{\text{TOB}}$ the pair
$$(c_u, \mathsf{NIZK}(a_u, r_u : c_u = E_{N,w_s}(a_u; r_u)).$$
2. Once $n-t$ valid pairs appear on the ledger, all parties compute $c_a = \prod_u c_u \bmod N^{s+1}$ where the product is over the u's that appeared. We have $c_a = E_{N,w_s}(a)$ where $a = \sum_u a_u \bmod N^s$.
Each member P_v of the next committee chooses random b_v, r_v, s_v. They compute $c_v = E_{N,w_s}(b_v; r_v)$ and $c'_v = c_a^{b_v} \cdot E_{N,w_s}(0; s_v) \bmod N^{s+1}$. They send to $\mathcal{F}_{\text{TOB}}$ the tuple:
$$(c_v, c'_v, \mathsf{NIZK}(b_v, r_v, s_v : c_v = E_{N,w_s}(b_v; r_v),\ c'_v = c_a^{b_v} \cdot E_{N,w_s}(0; s_v) \bmod N^{s+1}).$$
3. Once $n-t$ valid tuples appear on the ledger, all parties compute $c_b = \prod_v c_v \bmod N^{s+1}$ and $c_{ab} = \prod_v c'_v \bmod N^{s+1}$, where the product are over the v's that appeared. We have $c_b = E_{N,w_s}(b)$ where $b = \sum_v a_v \bmod N^s$.
And, because each c'_v contains $b_v a \bmod N^s$, we have $c_{ab} = E_{N,w_s}(ab \bmod N^s)$ where $d = ab \bmod N^s$.
All parties output (c_a, c_b, c_{ab}).

Fig. 30. The Triple protocol.

The Triple and Multiply protocols can be run in parallel as many times as we want. In the following, we use $c = \mathsf{Multiply}(c_1, \ldots, c_a)$ as shorthand for invoking Multiply an appropriate number of times on ciphertexts $c_1 = E_{N,w_s}(x_1), \ldots, c_a = E_{N,w_s}(x_a)$ to obtain $c = E_{N,w_s}(x_1 \cdot \cdots \cdot x_a \bmod N^s)$. This can be done in a standard tree structure and will then consume $\log a$ consecutive commitees, but one can also use the well-known Bar-Ilan and Beaver constant round technique to use only a constant number of committees. We will not go into details with this.

The Multiply protocol uses a decryption step when it consumes a triple. For this case, the decryption protocol (Fig. 9) is run without calling the RandomizeCiphertext subprotocol. We refer to this as DecryptNoRandomize. This is done because RandomizeCiphertext itself calls Multiply and we need to avoid circularity.

Protocol Multiply.
This protocol is parametrized by s, and will do secure multiplication on ciphertexts in $\mathbb{Z}_{N^{s+1}}$. The input consists of two ciphertexts $c_x = E_{N,w_s}(x), c_y = E_{N,w_s}(y)$

1. Let (c_a, c_b, c_{ab}) be the next available triple. All parties compute $c_\epsilon = c_x(c_a)^{-1} \bmod N^{s+1}$ and $c_\delta = c_y(c_b)^{-1} \bmod N^{s+1}$. Send c_ϵ, c_δ to the DecryptNoRandomize protocol for decryption.
2. Once the decryption results ϵ, δ are returned, all parties compute and output

$$c_{xy} = c_{ab} \cdot c_b^\epsilon \cdot c_a^\delta \cdot (N+1)^{\epsilon\delta} \bmod N^{s+1}.$$

It is straightforward to see that $c_{xy} = E_{N,w_s}(xy \bmod N^s)$.

Fig. 31. The Multiply protocol.

The first part of the Triple protocol where the random ciphertext c_a is created can be used stand-alone, and based on this we can do inversion using a well-known trick, also from Bar-Ilan and Beaver. Namely, given a ciphertext $c = E_{N,w_s}(x)$, let a committee create $c_a = E_{N,w_s}(a)$ for a random a, and then decrypt Multiply(c, c_a), to get $e = xa \bmod N^s$. Finally all players compute $c_a^{e^{-1}} \bmod N^{s+1} = E_{N,w_s}(x^{-1} \bmod N^s)$. We will refer to this ciphertext as Inverse(c).

Creating Random Encrypted Bits To make a random encrypted bit, we call the random oracle on input a label for this instance of the protocol, which produces an encryption of a random value x. The idea is now to compute securely a new encryption containing the Jacobi symbol $\left(\frac{x}{N}\right)$ of x modulo N which can easily be converted to a random bit. All parties compute the output ciphertext locally from data on the ledger. Once an encrypted bit is produced it is placed on an ordered list, and when we say in the following that we use the next available encrypted bit, we mean that all parties take the next unused ciphertext on the list. The protocol is found in Figure 32.

The required zero-knowledge proof in the RandBit protocol can be constructed from a Σ-protocol, which is turn a standard construction where the prover sends a random new pair d_u, d'_u containing a random number z_u and its Jacobi symbol. The prover gets a binary challenge and must either open d_u, d'_u, or open $a_u z_u \bmod N^s$ and $\left(\frac{a_u z_u}{N}\right)$. In the latter case, it must also be shown that $a_u z_u \bmod N^s$ is indeed the product of the plaintexts inside d_u and d'_u, and similarly for the Jacobi symbol. This is easy using standard techniques, where we can have a protocol with exponentially many challenges so this only adds a constant factor overhead.

Random Exponentiation Given a ciphertext $E_{N,w_1}(x)$, we want to produce a ciphertext $E_{N,w_1}(x^a \bmod N)$ where a is random exponent of specified bitlength κ bits. We can take from the next available encryptions of random bits $E_{N,w_1}(a_i)$, for $i = 0, \ldots, \kappa - 1$, and then follow a constant round protocol for secure exponentiation. This will allow us to use a constant number of committees for this operation. Our protocol is somewhat similar to the one from

Protocol RandBit.
This protocol is parametrized by s, and will create a random bit inside a ciphertext in $\mathbb{Z}_{N^{s+1}}$. The random oracle H is a here assumed to output a random value modulo N^{s+1}.

1. Let ℓ be a unique label for this instance of the protocol, and let $c_x = H(\ell)$, then for some x we have $c_x = E_{N,w_s}(x)$.
2. Each party P_u on the first committee assigned to this protocol instance will choose random a_u, r_u, r'_u and compute $c_u = E_{N,w_s}(a_u; r_u), c'_u = E_{N,w_s}((\frac{a_u}{N}); r'_u)$ and send to $\mathcal{F}_{\text{TOB}}$ the triple:
$$(c_u, c'_u, \mathsf{NIZK}(a_u, r_u, r'_u : c_u = E_{N,w_s}(a_u; r_u), c'_u = E_{N,w_s}(y_u; r'_u), y_u = (\frac{a_u}{N})).$$
3. Once $h = n - t$ valid triples $(c_{u_j}, c'_{u_j} \pi_{u_j})$ appear on the ledger, set $c_a = \mathsf{Multiply}(c_{u_1}, \ldots, c_{u_h})$ and $c'_a = \mathsf{Multiply}(c'_{u_1}, \ldots, c'_{u_h})$. Since the Jacobi symbol is multiplicative, we will have that $c_a = E_{N,w_s}(a), c'_a = E_{N,w_s}((\frac{a}{N}))$ for a random a.
4. Decrypt $\mathsf{Multiply}(c_x, c_a)$ to get $xa \bmod N^s$, and decrypt $\mathsf{Multiply}(c_x, c_a)$ to get $\sigma = (\frac{x}{N})(\frac{a}{N}) = (\frac{xa}{N})$. Therefore $c'_x := (c'_a)^\sigma \bmod N^{s+1} = E_{N,w_s}((\frac{x}{N}))$.
5. All parties compute and output
$$c_b = (c'_x \cdot (N+1))^{2^{-1} \bmod N^s} \bmod N^{s+1}.$$
This operation ensure that c_b will contain $((\frac{x}{N}) + 1)/2$ which is indeed a 0/1 value.

Fig. 32. The RandBit protocol.

[DFK+06], but is much easier to make maliciously secure in our context. It is found in Figure 33.

We will also need a protocol Exp for raising an encrypted number to a specific exponent. This is done exactly as RandEXP, except that we take an encrypted binary exponent $B_i = E_{N,w_1}(b_i),\ i = 0, \ldots, \kappa - 1$ as input instead of choosing it at random.

We will need this protocol to obliviously raise several different encrypted numbers to the same exponent, this can obviously be done by using the same encrypted bits in all instances of the protocol.

We will also need a protocol NewKey for generating a new key pair for a party, such that the secret key can be sent privately to that party, se Figure 34. The protocol therefore takes as input an encrypted one-time pad otp and outputs the secret encrypted under otp. For simpler exposition, the one-time pad is here assumed to consist of $2\log(N) := 2\delta$ random numbers in $\mathbb{Z}_N$. We one-time pad encrypt a bit x_i as $\mathsf{otp}_i + x_i \bmod N$, so decryption is obvious. Several obvious optimizations could be done to get a much more compact implementation.

J Implementing MPC

In this section, we describe how to add MPC capability to the Role Assignment Protocol. This is done almost entirely by using subprotocols we have already presented. We first explain an extension of the NewRole protocol, $\mathsf{NewRole}^{\text{MPC}}$: The original version outputs a randomized tag and a randomized public key belonging to some party P. The extended version also output a randomized encryption $E_{N,w_1}(\mathsf{otp}_P^{\text{OUT}})$, where $\mathsf{otp}_P^{\text{OUT}}$ was chosen by

Protocol RandEXP.
This protocol is parametrized by κ, the number of bits in the exponent to use. The input is $c = E_{N,w_1}(x)$

1. Each party P_u on the first committee assigned to this protocol instance chooses random y_u and $r_{u,i}, i = 0, \ldots, \kappa - 1$ and computes $c_{u,i} = E_{N,w_1}(y_u^{2^i} \bmod N; r_{u,i})$, for $i = 0, \ldots, \kappa - 1$. They send to $\mathcal{F}_{\text{TOB}}$ the tuple

$$(c_{u,0}, \ldots, c_{u,\kappa-1}, \mathsf{NIZK}((y_u, r_{u,0}, \ldots, r_{u,\kappa-1}) : c_{u,i} = E_{N,w_1}(y_u^{2^i} \bmod N; r_{u,i}),\ i = 0 \ldots \kappa - 1)$$

2. Once $h = n - t$ valid tuples appear on the ledger, say with indices $u_1, \ldots u_h$, then:
For each $i = 0, \ldots, \kappa - 1$, set $c_i = \mathsf{Multiply}(c_{u_1,i}, \cdots, c_{u_h,i})$. Setting $y = \prod_{j=1}^{h} y_{u_j} \bmod N$, we have that $c_i = E_{N,w_1}(y^{2^i} \bmod N)$.
3. Set $c' = \mathsf{Inverse}(c_0) = E_{N,w_1}(x^{-1} \bmod N)$. Send $\mathsf{Multiply}(c', c)$ to the $\mathsf{Decrypt}$ protocol.
4. Once the result z (which will equal to $xy^{-1} \bmod N$) is returned, all parties compute, for $i = 0, \ldots, \kappa - 1$ $d_i = c_i^{z^{2^i}} \bmod N^2$. We have that $d_i = E_{N,w_1}(x^{2^i} \bmod N)$.
5. Take the next κ available encrypted bits $B_i = E_{N,w_1}(b_i),\ i = 0, \ldots, \kappa - 1$. Set

$$X_i = \mathsf{Multiply}(B_i, d_i) \cdot (N + 1) \cdot B_i^{-1} \bmod N^2 = E_{N,w_1}(x^{b_i 2^i} \bmod N).$$

6. Finally, output $\mathsf{Multiply}(X_0, \ldots X_{\kappa-1})$ and $B_i = E_{N,w_1}(b_i),\ i = 0, \ldots, \kappa - 1$.

Fig. 33. The RandExp protocol.

Protocol $\mathsf{NewKey}(E_{N,w_1}(\mathsf{otp}_1), \cdots E_{N,w_{2\delta}}(\mathsf{otp}_\gamma)$.

1. Let $c_g = E_{N,w_1}(g)$ be an encryption with default randomness of the fixed generator $g \in \mathbb{Z}_N^*$. Run $\mathsf{RandExp}(c_g)$ twice with $\kappa = \delta$ to get ciphertexts $E_{N,w_1}(h), X_i = E_{N,w_1}(x_i),\ i = 1, \ldots, \delta$ and $E_{N,w_1}(\tilde{h}), \tilde{X}_i = E_{N,w_1}(\tilde{x}_i),\ i = 1, \ldots, \delta$.
Note that we have $h = g^x \bmod N$ and $\tilde{h} = g^{\tilde{x}} \bmod N$, where x is number with binary representation $x_1, \ldots x_\delta$ and similarly for $\tilde{x}$. So this is a key pair of the form we require.
2. Decrypt and output $h, \tilde{h}$. Also multiply the encryptions of the otp_i's by the encryptions of the bits in x and $\tilde{x}$, and output the results $E_{N,w_1}(\mathsf{otp}_1 + x_1 \bmod N), \ldots, E_{N,w_1}(\mathsf{otp}_\delta + x_\delta \bmod N)$, as well as $E_{N,w_1}(\mathsf{otp}_\delta + \tilde{x}_1 \bmod N), \ldots, E_{N,w_1}(\mathsf{otp}_{2\delta} + \tilde{x}_\delta \bmod N)$

Fig. 34. The NewKey protocol.

party P. The ciphertext is connected to the role, but cannot be connected to P because of the randomization.

We give some more details here on how we do the $\mathsf{NewRole}^{\mathrm{MPC}}$ protocol. The protocol is an extension of RandKey described in the main text.

1. We start from the assumption that, for some set of players $P_1, \ldots, P_{M'}$, we are given (gpk_i), for $i = 1, \ldots, M'$.
2. All players can now compute $H(\mathsf{b}, i)$ for $i = 1 \ldots M'$, where H is the random oracle, here assumed to output a random number in $\mathbb{Z}_N^*$. Here b is a unique batch number decouple the use of H in different batches. The protocol now considers a database with entries of form $(\mathsf{gpk}_i, E_{N,w_1}(\mathsf{otp}_i^{\mathrm{OUT}}), H(\mathsf{b}, i))$ and does a multiparty PIR protocol on this database similar to what we described in detail in the RandKey protocol. In the RandKey protocol, we did the PIR twice, with the same choice of random index t, once for each component of gpk_i. We now do it 4 times since we have a total of 4 components in each database entry. Having done what corresponds to Step 3 of RandKey we will have random encryptions $E_{N,w_1}(h_t), E_{N,w_1}(\tilde{h}_t), E_{N,w_1}(\mathsf{otp}_t^{\mathrm{OUT}}), E_{N,w_1}(H(\mathsf{b}, i))$.
3. In the final step, a randomized key rpk_t is produced from $E_{N,w_1}(h_t), E_{N,w_1}(\tilde{h}_t)$ and is output, exactly as in RandKey. Further $E_{N,w_1}(\mathsf{otp}_t^{\mathrm{OUT}})$ is output, and finally $\mathsf{tag}_t = H(t)^K \bmod N$ is output. This last output is produced as follows: K is a random number of length $\log_2(N)$, whose binary representation is given in encrypted form as set-up data. We can therefore do Exp on input $E_{N,w_1}(H(\mathsf{b}, i))$ and encrypted bits $E_{N,w_1}(K_j), i = 0, \ldots, \log_2(N) - 1$ to get $E_{N,w_1}(H(\mathsf{b}, i)^K)$ which is decrypted to get the output.

We will use $\mathsf{NewRole}^{\mathrm{MPC}}((\mathsf{gpk}_1, O_1), \ldots, (\mathsf{gpk}_{M'}, O_{M'})) = (\mathsf{rpk}_t, \mathsf{tag}_t, \mathsf{e}_{\mathsf{otp}^{\mathrm{OUT}}})$ as shorthand for a call to this protocol, where $O_P = E_{N,w_1}(\mathsf{otp}_P^{\mathrm{OUT}})$ is the encryption of the one-time pad provided by P.

Next, we describe how to evaluate a function f: We assume that f is given by an arithmetic circuit with addition and multiplication in $\mathbb{Z}_N$. We assume we are given encrypted inputs $c_{x_1=}E_{N,w_1}(x_1), \ldots, c_{x_a} = E_{N,w_1}(x_a)$. Starting from the inputs, parties compute addition by multiplying ciphertexts and multiplication by doing $c_3 = \mathsf{Multiply}(c_1, c_2)$ where c_1, c_2 contain the two plaintext values to be multiplied. We can allow random choices in the function by using RandBit to generate random encrypted bits. In the end we obtain the outputs on encrypted form: $c_{y_1=}E_{N,w_1}(y_1), \ldots, c_{y_b} = E_{N,w_1}(y_b)$.

First of all, inputs are supplied by parties in the beginning of a batch, where each party broadcasts information for the role assignment, each party P additionally broadcasts $c_P = E_{N,w_1}(x_P), \pi_P, c_{\mathsf{otp}_P^{\mathrm{OUT}}} = E_{N,w_1}(\mathsf{otp}_P^{\mathrm{OUT}})$, where π_P as usual is a zero-knowledge proof of plaintext knowledge for c_P, and $c_{\mathsf{otp}_P^{\mathrm{OUT}}}$ is an encrypted one-time pad to be used later for receiving output. P will delete his state except $\mathsf{otp}_P^{\mathrm{OUT}}$. We will use $\mathsf{Eval}(f, c_{x_1}, \ldots c_{x_a}) = (c_{y_1}, \ldots, c_{y_b})$ as shorthand for a call to this protocol.

Finally, we specify how to extend the role assignment protocol to allow for MPC, we do this by extending the OneBatch protocol from Figure 12 as follows, to get a protocol we call $\mathsf{OneBatch}^{\mathrm{MPC}}$:

1. In the Next Batch step, each party P broadcasts additionally an encrypted input $E_{N,w_1}(x_P)$ and an encrypted one-time pad $E_{N,w_1}(\mathsf{otp}_P^{\text{OUT}})$, both with zero-knowledge proofs of plaintext knowledge.
2. The calls to $\mathsf{NewRole}$ are replaced by $\mathsf{NewRole}^{\text{MPC}}$, with the effect than an encrypted one-time pad $\mathsf{e}_{\mathsf{otp}^{\text{OUT}}}$ is assigned to each role.
3. Let $c_{x_1}, \ldots c_{x_a}$ be the encrypted inputs that were delivered. Do $(c_{y_1}, \ldots, c_{y_b}) \leftarrow \mathsf{Eval}(\mathsf{f}, \mathsf{c}_{\mathsf{x}_1}, \ldots \mathsf{c}_{\mathsf{x}_a})$.
4. Given an encryption of the output $c_{y_j} = E_{N,w_1}(y_j)$ to be delivered to role j, use $\mathsf{e}_{\mathsf{otp}_j^{\text{OUT}}}$ to compute and output an encryption of y under $\mathsf{otp}_j^{\text{OUT}}$, as we did in the the NewKey protocol in Figure 34. This allows the party P who sent $\mathsf{otp}_j^{\text{OUT}}$ to get the output.

We define the protocol $\Pi_{\text{RA+MPC}}$ to be a number of sequential iterations of $\mathsf{OneBatch}^{\text{MPC}}$.

K Proofs of Security for the Role Assignment and MPC Protocol

In section J we specified a protocol $\Pi_{\text{RA+MPC}}$. The goal in this section is to show the following:

Theorem 8. *When for a constant c at most $T < M/(3+c)$ parties are adaptive corrupted and we set $n = \lambda$ then for a large enough constant* eno *we have that if* PEAS *is EUF-CMA, SO-IND-CPA, CSO-IND-CPA and CSO-ANON, then Π_{RA+MPC} securely implements $\mathcal{F}_{RA+MPC}^{\mathsf{PEAS},1/3,\mathbb{F},\gamma,m}$ in the $(\mathcal{F}_{SETUP}, \mathcal{F}_{TOB}, \mathcal{F}_{ATOMIC\text{-}SEND})$-model with a random oracle. Here $\mathbb{F}$ can be any class of functions with a bounded multiplication complexity over $\mathbb{Z}_N$ and γ and m can be any polynomial.*

We prove the theorem for $m = 1$ for notational convenience. The proof trivially adapts to $m > 1$.

We start by showing that the $\mathsf{Decrypt}$ and $\mathsf{Reshare}$ protocols in Figures 9, 10 produce correct outputs. Recall that there is a committee pair assigned to handle each batch of ciphertexts to decrypt. We maintain the invariant that when a pair is about to decrypt a batch, it holds $\mathsf{VSS}(d_S, \boldsymbol{v}_{d_S})$ with share bound 2^b, where d_S is the decryption exponent. This is ensured for the first pairs by $\mathcal{F}_{\text{SETUP}}$, and later by resharing d_S for the next committee pairs.

At all times, we will use α_i to denote the commitment computed from the public β_j's in the VSS, according to equation (2). This ensures that the i'th share of d_S satisfies $\alpha_i = \mathsf{Com}_{\mathsf{ck}}(\boldsymbol{v}_d[i]; v_i)$. Note that we have no issue of ciphertexts being ill-formed: any number is an encryption of some message.

Regarding the share bound after resharing, assume we use the specific sharing scheme mentioned in Section F, where the share matrix is of size polynomial in n, say we have at most n^a rows and columns, and where all entries in the sharing matrix, sweeping and recombination vectors are 1 or -1. Then, by inspection of the operations done, one obtains that if the share bound is 2^b before resharing, it will be $2^{2a \log n + b + 2k}$ after the resharing where k is the statistical security parameter. Namely, resharing a number with at most b bits first results in additive shares of size at most $b + k$ bits. These are now in turn shared using the threshold scheme. For this we need sharing vectors with $(b + k) + k = b + 2k$

bits entries, resulting in threshold shares with $a \log n + b + 2k$ bits[8]. These are combined using a recombination vector, which adds another $a \log n$ bits. This is already better than the approach using the integer version of Shamir's scheme, as here one gets a factor $n!$ multiplied on shares for each resharing, so that we add $\Omega(n \log n) + k$ bits to the share size for each resharing. We also sketched a protocol that even allows us to reduce the share size to a fixed amount.

For later use in the argument for security of our role assignment and MPC protocol, we want to argue that the decryption and reshare protocols work correctly. These proofs and many of the ones to follow make the following assumption:

Assumption HCNF (Honest committees, no forgeries): all committees used have honest majority and no signature of an honest party is forged.

If this assumption does not hold the protocol may, for instance, abort early or a corrupt player can send a message for a role she does not hold. It is shown elsewhere that under the security assumptions we make, HCNF holds with overwhelming probability, provided the global public keys and role keys are generated by an ideal functionality. This is of course not the case in the protocol. However, in a hybrid we construct later we can plug in ideally generated keys and conclude the assumption holds there. One now wants to say that if this failed in the protocol, we could distinguish, but on the other hand, indistinguishability depends on the assumption being true, so it seems the argument is circular. But this is not the case, as we use the first offending event to distinguish and up to that point, there is no difference. This is formalized in Lemma 1.

We now have:

Lemma 8. *Under HCNF, the invariant is maintained by protocol* Reshare *except with negligible probability. That is, the j'th committee doing decryption holds* $\mathsf{VSS}(d_S, \boldsymbol{v}_{d_S})$ *with share bound* $2^{(S+1)\log N + k + (j-1)(a \log n + 2k)}$.

Proof. The fact that the committee pair holds a VSS of d_S follows for the first pair from the fact that it receives the shares from the ideal functionality. For subsequent committee pairs, note that the reshare protocol only considers VSS's that pass the check in VSSverify, and the proof of Lemma 7 implies that, except with negligible probability, each such VSS all honest players in the receiving committee have shares of a well-defined value with a well-defined share vector. So such a VSS for player P_i is indeed of form $\mathsf{VSS}(z, \boldsymbol{v}_z)$ for some $z, \boldsymbol{v}_z$. Moreover, this value is equal to the original share s_i held by the player, as α_i defining that share is used as β_1^i in the VSS. The share bound follows immediately from the discussion above, and the fact that d_S it self is a number with at most $(S+1)\log N$ bits. □

Lemma 9. *Under HCNF, the decryption protocol outputs correct plaintexts except with negligible probability.*

Proof. We first consider a ciphertext c that is output by the initial call to RandomizeCipertext. Since the committee holds a VSS of the correct d_S, by Lemma 8, each α_i is a commitment

[8] Note that all players must prove in zero-knowledge that the shares they encrypt are in range, so corrupt players cannot force shares to be too large.

containing the correct i'th share s_i of d_S. Hence, by soundness of the zero-knowledge proof $\pi_{c,i}$, we can assume that, except with negligible probability, each decryption message $d_{c,i}$ that is delivered from the Gather protocol (and relayed by the threshold committee) is of form $d_{c,i} = c^{s_i} \bmod N^{s+1}$, where s_i is the corresponding additive share of d_S.

Consider now some fixed but arbitrary receiving party, and assume that there is some decryption message $d_{c,i}$ that this party did not receive in any of the messages coming from a set A of parties in the threshold committee. This means that all parties in A (claim they) did not get $d_{c,i}$ and so, for their last step message to be valid, they must have included a valid back-up message for $d_{c,i}$. We therefore have back-up messages $\{d_{c,i,j} |\ P_{u(j)} \in A\}$. Again by soundness of the zero-knowledge proofs, we can assume that $d_{c,i,j} = c^{s_j} \bmod N^{s+1}$ So, since A is a qualified set, by the properties of the reconstruction vector $\boldsymbol{r}^i_A$, it follows immediately that

$$\prod_{j, P_{u(j)} \in A} d_{c,i,j}^{\boldsymbol{r}^i_A[j]} = d_{c,i} = c^{s_i} \bmod N^{s+1} \ ,$$

and so we can conclude that the final product computed satisfies

$$\prod_{i=1}^{n} d_{c,i} = c^{\sum_{i=1}^{n} s_i} = c^{d_S} \bmod N^{s+1}$$

resulting in correct decryption of c.

Considering the randomization step, recall that it outputs

$$c = \bar{c} \cdot \mathsf{Multiply}(c^*, H(\bar{c}, R)) \bmod N^{s+1} \ ,$$

where $\bar{c}$ is the original input ciphertext. The Multiply protocol relies for correctness on the decryption protocol without the randomization, but this is what we just proved is correct. By the homomorphic property, we can therefore assume that the plaintext contained in c is of form $m + \alpha \cdot \beta \bmod N^s$, where m, α, β are the plaintexts contained in $\bar{c}, H(str)$ and c^*, respectively. In the set-up for the protocol, c^* is generated such that $\beta = 0$, and so we conclude that c also contains m and the decryption is correct. □

Main idea for proof of security We now outline the idea for the proof of security of the global role assignment protocol: We first show a UC simulator UCsim, it will do a straight-line simulation (as required for UC) which is possible because it knows the factorization of N and hence can do most of its job by simply following the protocol. The only caveat occurs when the protocol decrypts an output corresponding to an output from $\mathcal{F}_{\mathrm{RA+MPC}}$, such as a public key for a role. In such a case, the ciphertext produced in the simulation cannot be assumed to contain the right value. UCsim fixes this by changing the ciphertext c^* from the set-up such that it contains 1 instead of 0. This allows it to engineer the randomization step in the decryption such that the "randomized" ciphertext that is actually decrypted contains the correct value. Therefore, at the end of the day, the only difference between simulation and real protocol is that some of the Paillier ciphertexts that are never decrypted contain different values in the two cases.

However, since UCsim knows the factorization of N (it gets it from $\mathcal{F}_{\text{RA+MPC}}$), we cannot directly appeal to CPA security of Paillier to say that this difference cannot be detected. Instead, we exploit the fact that when proving indistinguishability of simulation and protocol, we are no longer doing UC simulation, so rewinding is allowed. We show that, even without the factorization of N, we can emulate both the real protocol and the simulation using rewinding and a variant of the single inconsistent player (SIP) technique. This way, we define two processes called ProtRewind and SimRewind producing views for the environment that are perfectly indistinguishable from the protocol and from the simulation, respectively. Note the in doing this we are rewinding the environment. This is allowed as we do it as a proof technique. The UC simulator itself is straight line. Finally, skipping many details, we use the fact that the SIP technique allows us to simulate decryption without knowing the secret Paillier key and the computational assumptions we make to argue that ProtRewind and SimRewind are computationally indistinguishable.

The UC Simulator The high-level approach of UCsim, shown in Figure 35, is standard: emulate the honest parties by following the protocol, and when output is generated or a new party is corrupted, adjust the internal state so it matches what the functionality requires. We will use the variant of the UC framework where there is no explicit adversary, and the environment Z acts also as adversary.

Towards understanding the simulator, we note a few points: decryption of Paillier ciphertext occurs in two cases: one case is when a ciphertext is decrypted as a part of the multiplication subprotocol, where we consume a multiplication triple, or when the decryption corresponds to a private output for an honest player. The other case is called an *output decryption* where decryption occurs, corresponding to an output that $\mathcal{F}_{\text{RA+MPC}}$ leaks to the simulator; either a public output or a private output for a corrupt player. Here, the result of the decryption is dictated by $\mathcal{F}_{\text{RA+MPC}}$, and the simulator takes measures to ensure that the correct output is generated.

When an honest player is corrupted, the player may possess several different types of data as listed below. Note that, in general, a player always deletes data that is no longer needed, so the only data found in memory are the latest private keys received and possibly data pertaining to the current (unexecuted) role of the player.

- *Private output:* the player may hold private output from the MPC.
- *Private keys:* the player may hold some private keys, corresponding to public keys it has been assigned. The key may be *external*, i.e., it is a key produced as output from $\mathcal{F}_{\text{RA+MPC}}$, or it can be *internal*, i.e., it is produced only for the purpose of executing the protocol.
- *Shares:* the player may hold shares of the Paillier decryption key.
- *One-time pad:* the player may hold a one-time pad, if it is waiting for private output, either a private key or private output from the MPC.

It can be seen in the simulation that the simulator handles corruptions in a very simple way: it has a simulated state st_i for the corrupted player P_i containing simulated output that the player has, and has not yet been deleted. Now, the simulator receives the correct output

Simulator UCsim for the Role Assignment protocol.

Intialize. Receive N and the factors of N from $\mathcal{F}_{\text{RA+MPC}}$. Use this to emulate $\mathcal{F}_{\text{SETUP}}$, with one adjustment: the ciphertext c^* is generated as an encryption of 1 instead of 0. Initialize a copy of all honest players and give them the simulated set-up data from $\mathcal{F}_{\text{SETUP}}$. Send all private set-up data meant for corrupted players to Z. Initialize a copy of $\mathcal{F}_{\text{TOB}}$.

Main Process Execute the code of the honest players according to the protocol (while Z plays for the corrupted players). The interface of the internal emulation of $\mathcal{F}_{\text{TOB}}$ is connected to Z and the honest players as in the real protocol.

- The random oracle H is emulated using standard lazy sampling of random values, however, certain inputs are handled in a special way as detailed below.
- Whenever a corrupt player supplies a ciphertext, it is always accompanied by a zero-knowledge proof, from which the simulator straight-line extracts the corresponding secret data used to generate the message. As a result, the simulator knows plaintext and randomness for all ciphertexts in the global state, as well as all shares of the secret Paillier key held by the committees.
- When an output decryption occurs, execute the decryption subroutine below.
- When a corruption occurs, execute the corruption handling subroutine below.

Output Decryption Let $\bar{c} \in \mathbb{Z}^*_{N^{s+1}}$ be the ciphertext to be decrypted, and let z be the output generated by $\mathcal{F}_{\text{RA+MPC}}$. Let m be the plaintext contained in $\bar{c}$, which most likely is different from z. To fix this, the simulator does the following:

1. Let L be the label of the relevant batch of ciphertexts to decrypt. The simulator chooses R at random and programs $H(L)$ to be an encryption of R. Once the batch appears on the ledger, the simulator programs $H(\bar{c}, R)$ to be a random encryption of $z - m$. If the random oracle has been called before with an input containing R, the simulator fails and aborts.
2. Since c^* contains 1, the output $c = \bar{c} \cdot \mathsf{Multiply}(c^*, H(\bar{c}, str)) \bmod N^{s+1}$ from RandomizeCiphertext contains z. The simulator can therefore let the rest of the decryption proceed normally.

Corruption Handling When a player P_i is corrupted, the simulator is given whatever output out_i that $\mathcal{F}_{\text{RA+MPC}}$ has given to P_i and has not yet been deleted. The simulator also has the internal state st_i of P_i as generated in the simulation so far. Note that st_i will contain output values generated for P_i in the simulation. The simulator replaces these values by out_i and hands the resulting state to Z.

Fig. 35. The UCsim simulator.

out_i from $\mathcal{F}_{\mathrm{RA+MPC}}$, and it simply replaces the simulated output in st_i by out_i and hands the resulting state to Z. Let us explain intuitively why this does not create any inconsistencies that Z could use to tell it is in the simulation: In the real protocol, P_i gets output by first broadcasting an encrypted one-time pad otp_i, and then later the value $\mathsf{out}_i + \mathsf{otp}_i \bmod N$ is decrypted in public, allowing P_i to compute out_i. In the simulation, some random value rnd_i was decrypted, so when the simulation claims that the output was out_i, it implicitly claims that the original one-time pad was $\mathsf{rnd}_i - \mathsf{out}_i \bmod N$. This is most likely not the value used for the simulated encryption of the pad, but Z cannot detect that the claim is false, as it only knows the encryption of the pad, and the randomness for the encryption has been deleted by P_i by construction.

Finally, the simulator can fail, if it is not able to program the oracle as needed. Intuitively, this should not happen, as the programming is done before R is decrypted, so the environment would have to guess R from $E_{N,w_s}(R)$ to make an offending call. As the secret Paillier key is used in the simulation we cannot immediately argue that failure happens with negligible probability, but we will do so later, in a hybrid that is shown indistinguishable without assuming that offending calls are unlikely.

As a consequence of these observation and Lemma 9, we get:

Lemma 10. *Under HCNF, the Π_{RA+MPC} protocol and the protocol instance run by* UCsim *produce correct outputs, except with negligible probabilty.*

Proof. In the protocol, one sees by simple inspection of the subprotocols, that as long as the decryption produces correct results, each subprotocol works correctly. This is because they all consist of homomorphic evaluation on ciphertexts, inputs supported by zero-knowledge proofs of plaintext knowledge and decryption. Further, the supply of random bits comes from the random oracle, so is correctly distributed. For the simulation, UCsim engineers the output ciphertexts so they contain the correct values, and Lemma 9 guarantees that these values are actually decrypted. □

Processes. In the following, a *process* is an algorithm that runs the environment Z is its head, as well as some number of other parties. The output of a process is whatever Z outputs. The Protocol process runs the protocol composed with Z and the resource functionalities the protocol requires, random oracle H, $\mathcal{F}_{\mathrm{SETUP}}$ and $\mathcal{F}_{\mathrm{TOB}}$. The Simulation process is the standard ideal process in the UC framework, it runs UCsim composed with Z and $\mathcal{F}_{\mathrm{RA+MPC}}$.

We first define $\mathsf{Simulation}^{\mathrm{ZKSIM}}$ and $\mathsf{Protocol}^{\mathrm{ZKSIM}}$, which are exactly the same as Simulation and Protocol, respectively, except that all ZK proofs done by honest parties are simulated.

We proceed to define two new processes $\mathsf{ProtRewind}(d_S)$ and $\mathsf{SimRewind}(d_S)$. Both take the secret Paillier key d_S as input.

To give a precise description, we recall that both simulation and protocol proceed in *batches*: in a batch, the first step is that some parties may give input to whatever secure computation is specified for the batch. Then, the secure computation is done, and new keys to be used in future batches are generated. After inputs have come in, the execution of a batch proceeds in a number of *epochs*. An epoch starts at the time a committee reshares the secret Paillier key for a number of committees, or more precisely, at the time where the first

honest party in the resharing committee starts executing the Reshare protocol. Note that the first epoch will not start until all inputs for the batch have been specified. Some committee C, that receives shares of the secret key, is assigned to do resharing in the next epoch. The epoch ends when the first member of C starts executing the Reshare protocol. This will not happen until it can be seen on the ledger that all committees in the epoch have done their work. *We assume throughout that each epoch involves a constant number of committees.*

As a final prerequisite, note that there is an initial committee pair C_0 that receives from $\mathcal{F}_{\text{SETUP}}$ a VSS of the secret Paillier key d_S. $\mathcal{F}_{\text{SETUP}}$ is executed in both protocol and simulation (with a change in the simulation that is irrelevant here). The ProtRewind(sec) process can be found in Figure 36.

Process ProtRewind(sec).

Intialize. It is assumed that sec is a number chosen from the same domain as the secret Pailier key d_S. Emulate $\mathcal{F}_{\text{SETUP}}$, with one modification: Choose a valid sharing vector $\boldsymbol{v}_{sec}$ and give VSS($sec, \boldsymbol{v}_{sec}$) to C_0. Now, execute the protocol composed with Z and random oracle H, as specified, except that each epoch is done differently, as specified below. For every zero-knowledge proof from a corrupt player, extract on-line the witness used. As a result of this, and because the actions of honest players can be observed, the process knows the content of every Pailler ciphertext and every VSS share computed in the protocol.

Epochs execution. We specify how each epoch is done:

1. At the start of the epoch, let the assigned committee execute the Reshare protocol as specified. For each committee pair (C_{add}, C_{th}) receiving a VSS in this epoch, where C_{add} is the additive committee and C_{th} is the threshold one, do as follows:
   (a) Pick a random member of C_{add} to be the single inconsistent player (SIP).
   (b) When the committee pair does decryption of a batch of ciphertexts, execute the Decrypt protocol as is, except that the decryption messages of the SIP are computed in a special way. Let c be a ciphertext to decrypt after the randomization step (see Figure 9), and note that the decryption result m is known. Let P_u be the SIP. Let the (correctly computed) decryption messages of the other parties in C_{add} be $\{d_{c,i} |\ i \neq u\}$ and set

$$d_{c,u} = (N+1)^m \prod_{i \neq u} d_{c,i}^{-1} \bmod N^{s+1}.$$

   Let P_u send the following decryption message using the Gather protocol:

$$d_{c,i}, \pi_{c,u},$$

   where $\pi_{c,u}$ is a simulated proof.
2. If at any point, the SIP in some additive committee is corrupted, or is not in the core set after the Gather protocol is done, rewind the entire process to the state it had at the start of the epoch, and go to step 1. If the epoch ends without rewinding happening, consider the set of honest parties that are in the core set after Gather (since we assume honest majority in committees, and the core has size at least $n - t$, there must be at least 1 such party). Select a random party P in this set. If P is not the SIP, rewind.
   If no rewind took place, continue to the next epoch.

Fig. 36. The ProtRewind process.

We then define the SimRewind(sec) process. We get it by modifying the simulation in exactly the same way as we modified the protocol to get ProtRewind(sec). More precisely,

UCSim works by directly executing the protocol. In SimRewind(sec), we will modify the execution of each epoch to use rewinding exactly as in ProtRewind.

By various lemmas proved below, we will first be able to conclude the following

$$\mathsf{Protocol} \approx_s \mathsf{Protocol}^{\text{ZKSIM}} \approx_p \mathsf{ProtRewind}(d_S)$$

$$\mathsf{SimRewind}(d_S) \approx_p \mathsf{Simulation}^{\text{ZKSIM}} \approx_s \mathsf{Simulation}\ ,$$

where $\approx_p$ denotes perfect indistinguishability and $\approx_s$ denotes statistical indistinguishability. And then, using the computational assumptions we make we will, via a number of intermediate hybrids, conclude that $\mathsf{ProtRewind}(d_S) \approx_c \mathsf{SimRewind}(d_S)$, which implies the result we want. Here $\approx_c$ denote computational indistinguishability.

Lemma 11. *We have*

$$\mathsf{Protocol} \approx_s \mathsf{Protocol}^{ZKSIM},$$

$$\mathsf{Simulation}^{ZKSIM} \approx_s \mathsf{Simulation}.$$

Proof. This is immediate by statistical zero-knowledge of the proofs we use – even under adaptive corruption, as an honest party always deletes randomness and witness immediately after sending its single message. □

Lemma 12. *Under HCNF, we have*

$$\mathsf{Protocol}^{ZKSIM} \approx_p \mathsf{ProtRewind}(d_S),$$

$$\mathsf{SimRewind}(d_S) \approx_p \mathsf{Simulation}^{ZKSIM},$$

and each of the two processes run in expected polynomial time.

Proof. We argue that $\mathsf{Protocol}^{\text{ZKSIM}} \approx_p \mathsf{ProtRewind}(d_S)$: Note that the only difference between the two processes is that the decryption step is executed differently in the two cases, while the output plaintext is always the correct one that is actually contained in the ciphertext to decrypt.

Moreover, note that when $\mathsf{ProtRewind}(d_S)$ creates an execution of an epoch, it has exactly the same distribution as in $\mathsf{Protocol}^{\text{ZKSIM}}$. The decryption message from the SIP is computed in a different way, but its distribution remains the same, namely we maintain, even in the rewinding process that the decryption messages for ciphertext c and plaintext m are computed correctly from the corresponding shares for all parties except the SIP, and then the SIP's message is fixed by the equation $\prod_i d_{c,i} = (N+1)^m \bmod N^{s+1}$. Note that we do not get the the decryption messages from the parties when they send them, which would give a problem with rushing. We compute them from the corresponding shares, which are known as we know the contents of all encryption sent by corrupted parties. We are merely computing the correct decryption messages for the SIP in an indirect way from the result and the share of all other parties. This will perfectly give the same decryption message. In particular, this means that Z has no information on which party we choose as the SIP, all parties in C_{add} behave the same way in the view of Z. This in turn implies that all decisions that lead to the

choice of the player P at the end of an epoch are taken independently of the choice of SIP. Therefore, the probability that we do not rewind and keep the view for Z that we generated is exactly $1/n$, and the decision to rewind or not is independent of the actual view. This, and the fact that each view we make going forward has the right distribution, implies the first conclusion.

The second conclusion follows by essentially the same argument, we leave the details to the reader.

Finally, the claim on the expected run time follows from the fact that everything in the protoocol is polynomial time so the only question is regarding the expected number of time we rewind. If there is a constant c number of committees in an epoch, then since each decryption avoids rewinding with an independent probability $1/n$, the probability we get a complete simulation of an epoch is $1/n^c$ and hence the expected number of rewinds is polynomial, n^c. □

We now define a two new processes $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$, $\mathsf{SimRewind}^{\mathrm{ZRND}}(d_S)$ which are "zeroing the randomness" used in the MPC. They are the same as $\mathsf{ProtRewind}(d_S)$ and $\mathsf{SimRewind}(d_S)$, respectively except for the following:

1. Let $\boldsymbol{b}$ be the string of random bits used for generating the new keys $\mathsf{gpk}, \mathsf{gsk}$ and rpk_i, in some batch. Thus, $\boldsymbol{b}$ determines the secret and public keys generated, the random indices used in the PIR protocols, and the random exponents used for randomizing role keys. The process selects $\boldsymbol{b}$ at random, but programs the random oracle such that the bit used in the protocol is always 0. This is done by letting the oracle output for each such bit $E_{N,w_s}(x)$ where x is a random number of Jacobi symbol -1 mod N.
2. The process computes the set of keys $\mathsf{gpk}, \mathsf{gsk}, \mathsf{rpk}$ that would result from $\boldsymbol{b}$. It runs the protocol exactly as $\mathsf{ProtRewind}(d_S)$ ($\mathsf{SimRewind}(d_S)$) does, except when decrypting a Paillier ciphertext c that is supposed to contain information about a key in $\mathsf{gpk}, \mathsf{gsk}, \mathsf{rpk}$. For such a ciphertext it does the following: Let m' be the plaintext that is consistent with $\mathsf{gpk}, \mathsf{gsk}, \mathsf{rpk}$. Ignore the content of c and when computing the decryption message of the SIP, set
$$d_{c,u} = (N+1)^{m'} \prod_{i \neq u} d_{c,i}^{-1} \bmod N^{s+1}.$$
3. $\boldsymbol{b}$ contains, in particular, the indices used for sampling random parties when the roles are determined. It therefore also determines which executions of the PIR protocol would lead to collisions. The process uses the PRF key K to compute the tag that would indicate the collision and will force decryption of that tag, as described in the previous step.

Intuitively, $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$ removes from the process all information about how $\mathsf{gpk}, \mathsf{gsk}, \mathsf{rpk}$ were generated, but still tries to make it seem like a correctly generated set of keys were produced.

In the proof of the following lemma, we will use a PPBox specified in Figure 37. It is designed to allow emulation of the resharing and decryption steps in the protocol. Towards understanding its specification: a VSS $\mathsf{VSS}(sec, \boldsymbol{v}_{sec})$ consists of commitments to the secret sec and the randomness for the sharing (found in the vector $\boldsymbol{v}_sec$). A non-interactive VSS

PPBox(sec).
This PPBox gets a master secret sec as input and will now compute and output a series of VSSs following the pattern of our Reshare protocol. Each time it is called (by an adversary) it will receive an auxiliary input that it will interpret as a command, sometimes with an extra input for executing the command. It may return sets of shares, but also other information as auxiliary output.

Intialize. When asked to initialize, it will get a commitment public key ck and names of the members of an initial committee pair C_0 as auxiliary input. It computes $\mathsf{VSS}(sec, \boldsymbol{v}_{sec})$, and returns as auxiliary output the commitments from the VSS. It sets $\mathsf{ReshareVSS} = (\mathsf{VSS}(sec, \boldsymbol{v}_{sec}), C_0)$, and marks all members of C_0 as honest (so far).
In the following, we always let C denote the committee pair in ReshareVSS, and $\mathsf{VSS}(sec, \boldsymbol{v}_{sec})$ denotes the VSS held by C.

Reshare When asked to reshare to a committee pair R, for each share s_i belonging to an honest member of the threshold committee in C, compute $\mathsf{VSS}(s_i, \boldsymbol{v}_{s_i})$, return each share s_i and opening information for the commitment α_i to s_i in the VSS. Return as auxiliary output the commitments from the VSS. Note that in the game, the shares returned will be encrypted under the receivers' public keys before being returned to the adversary.

Recombine When asked to recombine for a committee pair R, receive a qualified set A chosen from the threshold committee in C. For each corrupted member P of A and each s_i belonging to P, get $\mathsf{VSS}(s_i, \boldsymbol{v}_{s_i})$ as well as all shares inside that VSS and opening information for all commitments in the VSS. Note that the box has the same information for the honest members of A as it created VSS for those on its own. Compute a linear combination over the VSSs from A exactly as in the recombination step in the Reshare protocol and store the resulting $\mathsf{VSS}(sec, \boldsymbol{v}_{sec})$.

Decrypt On command Decrypt 1 and input ciphertext c, committee pair R and the name of an SIP party from the additive committee in R, compute decryption messages $d_{c,i}$ as in the Decrypt protocol for all honest parties in the additive committee except the SIP, and return the $d_{c,i}$'s.
On command Decrypt 2 and input ciphertext c, committee pair R and SIP name, compute backup messages as in the Decrypt protocol for all honest parties in the threshold committee, backing up all parties in the additive committee except the SIP, in the notation from the Decrypt protocol, all $d_{i,j,c}$ where $i \neq SIP$ and s_j belongs to an honest party. Return the backup messages computed.

End of Epoch On command End of Epoch and input some committee pair R that was reshared to earlier, set $\mathsf{ReshareVSS} = (\mathsf{VSS}(sec, \boldsymbol{v}_{sec}), R)$. Note that box indeed holds a VSS for the committee pair.

Corrupt On command Corrupt and the name of a party P, then for all VSS's where this party was a receiver of shares return all shares and corresponding opening information for the commitments. Mark P as corrupt.

Fig. 37. The PPBox we use.

message for a secret s is computed from $\mathsf{VSS}(s, \boldsymbol{v}_s)$ and consists of the commitments (to secret and randomness) and for every share s_i, a ciphertext that contains s_i encrypted for the receiver of s_i and also a ciphertext containing opening information for a commitment α_i to s_i. α_i can be computed from the commitments in the VSS. Finally, the VSS message contains zero-knowledge proofs that everything was computed correctly.

It is straightforward to verify that it is a box with privacy structure that contains a set in every involved committee pair that is unqualified in the secret sharing scheme we use. Namely: the shares of corrupt parties is statistically independent of the master secret *sec*, the commitments returned are statistically hiding, and the decryption and backup message returned for decryption only depend on an unqualified set of shares.

Lemma 13. *Under HCNF, and assuming PEAS is CSO-IND-CPA secure we have*

$$\mathsf{ProtRewind}^{ZRND}(d_S) \approx_c \mathsf{ProtRewind}(d_S),$$

$$\mathsf{SimRewind}^{ZRND}(d_S) \approx_c \mathsf{SimRewind}(d_S).$$

Proof. We will show that $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S) \approx_c \mathsf{ProtRewind}(d_S)$, the proof of the other conclusion is essentially the same. We will use the CSO-IND-CPA game with the PPBox specified in Figure 37. This box will hold the secret Paillier key d_S.

We define a process Hybrid that interacts with Z and also plays the CSO-IND-CPA game. It runs in the same way as $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$, with the following modifications:

1. Instead of selecting $\boldsymbol{b}$ itself, it gets the keys $\mathsf{gpk}, \mathsf{gsk}, \mathsf{rpk}$ for the current batch from the game, and asks in the game for a leakage function f defined as follows: by definition in the game, f knows the random bits used for generating the keys. For each such bit b_i, it encodes the bit as a random number $X_i \in \mathbb{Z}_N^*$ of Jacobi symbol $(-1)^{1-b_i}$ and returns an encryption $E_{N,w_1}(X_i)$ [9]. Hybrid programs the random oracle such that $E_{N,w_1}(X_i)$ is used in the protocol when the i'th random bit is generated.
2. For decryption of ciphertext c, do as in $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$, except that we send Decrypt 1 and Decrypt 2 commands to PPBox to get the appropriate decryption and backup messages for c. The process adds simulated zero-knowledge proofs as required to match the format sent in the protocol.
3. For each call to Reshare, make a call to the PPBox in the game to get each of the VSS's sent by honest players. For each such VSS from player P, also get encryptions from the game of each share s_i and of opening information for the commitments α_i to s_i. Put this together with simulated zero-knowledge proofs to get a complete VSS message and send it on behalf of P. When enough VSS messages from a set A of parties are delivered by $\mathcal{F}_{\mathrm{TOB}}$, extract the shares and opening information for commitments from the VSSs by corrupt players in A and send these to the PPBox together with A. The PPBox will do the recombination to get a new sharing of d_S for the receiving committee pair.
   When a player P on a committee is corrupted, send a corrupt P message to the game to get gsk_P and also make a call to PPBox to get the share of P and opening information for the commitments it made in the VSSs. Return this to Z in response to the corruption.

[9] In some cases, one actually wants $X_i \in \mathbb{Z}_{N^s}^*$ for some s, but we ignore this here for simplicity.

Once the process has ended, output the bit that Z outputs.

It is now easy to verify that if the game is played when the secret bit to guess is 1, we get a process that is perfectly indistinguishable from $\mathsf{ProtRewind}(d_S)$, while if the bit is 0 we are perfectly indistinguishable from $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$. Namely, in both cases the VSSs occurring in the resharing steps and the decryption messages are perfectly emulated and the secret bit exactly switches between using the real randomness or zeros. Hence Z's distinguishing advantage equals the advantage we have in winning the game, which is negligible by assumption, as we do corruptions in the privacy structure of the PPBox by the HCNF premise. So the lemma follows. □

We define $\mathsf{ProtRewind}^{\mathrm{ZRND}}(0)$ to the the same as $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$, except that d_S is replaced by 0. $\mathsf{ProtRewind}^{\mathrm{ZRND}}(0)$ is defined similarly.

Lemma 14. *Under HCNF, and if PEAS is SO-IND-CPA secure we have*

$$\mathsf{ProtRewind}^{ZRND}(d_S) \approx_c \mathsf{ProtRewind}^{ZRND}(0),$$

$$\mathsf{SimRewind}^{ZRND}(d_S) \approx_c \mathsf{SimRewind}^{ZRND}(0).$$

Proof. We show that $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S) \approx_c \mathsf{ProtRewind}^{\mathrm{ZRND}}(0)$, the proof of the other part is essentially the same. We define a process that interacts with Z and also plays the SO-IND-CPA game, where the PPBox (the same as in the previous proof) is given d_S or 0 initially. It operates exactly as $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$, except that it uses calls to PPBox to emulate the VSSs in resharing, and the decryption messages, exactly as we did in the proof of the previous lemma, and it does not compute the keys $\mathsf{gpk}, \mathsf{gsk}, \mathsf{rpk}$ itself, but gets them from the game. When the process ends, output what Z outputs.

Note that the process no longer contains any information on the random bits used for generating keys, and therefore Z cannot detect that the keys are received from the game and not generated by the process. Therefore, if the PPBox holds d_S, we are perfectly emulating $\mathsf{ProtRewind}^{\mathrm{ZRND}}(d_S)$, and if it holds 0 we perfectly emulate $\mathsf{ProtRewind}^{\mathrm{ZRND}}(0)$. Hence Z's distinguishing advantage equals the advantage we have in winning the game, which is negligible by assumption. The lemma follows. □

For the next two lemmas, we note that CPA security of Paillier encryption clearly follows from CSO-IND-CPA security, which we assume throughout.

We define a modification of $\mathsf{SimRewind}^{\mathrm{ZRND}}(0)$: $\mathsf{SimRewind}^{\mathrm{ZRND,PROT}}(0)$, where we make the process look more like the protocol: we set the ciphertext c^* from the set-up used in the RandomizeCiphertext protocol so it contains 0 as in the protocol and we drop the programming of the random oracle that UCSim uses.

Lemma 15. *Under HCNF and CPA security of Paillier, we have*

$$\mathsf{SimRewind}^{ZRND}(0) \approx_c \mathsf{SimRewind}^{ZRND,PROT}(0).$$

Moreover an offending call to the random oracle that would make UCSim *fail, occurs with negligible probability in* $\mathsf{SimRewind}^{ZRND}(0)$.

Proof. Follows immediately by CPA security of Paillier, as the only effect of the changes occur inside Paillier ciphertexts that we can control by programming the setup or the random oracle. □

From the above lemma and the previous ones, we can now conclude that the offending call also must occur with negligible probability in $\mathsf{Simulation}$ (and the intermediate processes). By Lemma 1 this means that it holds in all hybrids that there is a negligible probability that there are offending calls to the random oracle.

We define processes $\mathsf{ProtRewind}^{\mathrm{ZRND}}_{\mathrm{LOSSY}}(0)$ and $\mathsf{SimRewind}^{\mathrm{ZRND,PROT}}_{\mathrm{LOSSY}}(0)$ where we replace the ciphertext w_S from the set-up by a ciphertext containing 0. This means that all ciphertexts generated by parties or in set-up are lossy, i.e., they actually contain 0, instead of the plaintext the party had in mind. Note, however, that it is still possible to extract witnesses from zero-knowledge proofs, showing how a ciphertext was formed, and hence which plaintext the party had in mind.

Lemma 16. *Under HCNF and CPA security of Paillier, we have*

$$\mathsf{SimRewind}^{ZRND,PROT}(0) \approx_c \mathsf{SimRewind}^{ZRND,PROT}_{LOSSY}(0),$$

$$\mathsf{ProtRewind}^{ZRND}(0) \approx_c \mathsf{ProtRewind}^{ZRND}_{LOSSY}(0).$$

Proof. Follows immediately from CPA security of Paillier, as the only effect of the changes occur inside a Paillier ciphertext from the setup. □

In both $\mathsf{ProtRewind}^{\mathrm{ZRND}}_{\mathrm{LOSSY}}(0)$ and $\mathsf{SimRewind}^{\mathrm{ZRND,PROT}}_{\mathrm{LOSSY}}(0)$, the ciphertexts from the set-up that would contain the PRF key K now contain 0. We can therefore replace the tags computed from K by random values and force decryption of these random tags at the end of every PIR protocol execution. Since there is no information on K available in the view of Z, we can use PRF security to argue that this change cannot be detected. Naming the new processes $\mathsf{SimRewind}^{\mathrm{ZRND,PROT}}_{\mathrm{LOSSY,RNDTAG}}(0)$ and $\mathsf{ProtRewind}^{\mathrm{ZRND}}_{\mathrm{LOSSY,RNDTAG}}(0)$, we conclude that

Lemma 17. *Under HCNF, we have*

$$\mathsf{SimRewind}^{ZRND,PROT}(0) \approx_c \mathsf{SimRewind}^{ZRND,PROT}_{LOSSY,RNDTAG}(0),$$

$$\mathsf{ProtRewind}^{ZRND}(0) \approx_c \mathsf{ProtRewind}^{ZRND}_{LOSSY,RNDTAG}(0).$$

Proof. Note that the function we use to compute tags $x \mapsto H(x)^K$ is a secure PRF under the DDH assumption which clearly follows from the SO-IND-CPA security we assume throughout. Hence a distinguisher contradicting the lemma would contradict SO-IND-CPA security by the reduction we just sketched. □

In both $\mathsf{SimRewind}^{\mathrm{ZRND,PROT}}_{\mathrm{LOSSY,RNDTAG}}(0)$ and $\mathsf{ProtRewind}^{\mathrm{ZRND}}_{\mathrm{LOSSY,RNDTAG}}(0)$, all information about how new keys are generated has been removed. In the view of Z, the situation is therefore equivalent to the keys being generated by the ideal functionality. We can therefore conclude that assuming SO-ANON security, we will have honest majority in all commitees except with

negligible probability, and no signature from an honest party will be forged, assuming CMA security of the signature scheme. So, under the complexity assumptions we made, the HCNF assumption holds in these processes, and by Lemma 1, we conclude that it also holds in all other processes, except with negligible probability.

It therefore follows by Lemma 10 that the original Protocol process generates correct outputs and this is preserved over the processes we have derived from the protocol. The same can be concluded for the processes derived from Simulation. We conclude that the output generated by $\mathsf{SimRewind}^{\text{ZRND,PROT}}_{\text{LOSSY,RNDTAG}}(0)$ and $\mathsf{ProtRewind}^{\text{ZRND}}_{\text{LOSSY,RNDTAG}}(0)$ have indistinguishable distributions, i.e., indistinguishable from correctly generated keys $\mathsf{gpk}, \mathsf{gsk}, \mathsf{rpk}$ and correct outputs from the MPC. Further, even though Simulation and processes derived from it use dummy inputs for honest players to the MPC, this cannot be detected once ciphertexts are lossy. Also observe that, except for the choice of inputs, the protocol is executed in exactly the same way in $\mathsf{SimRewind}^{\text{ZRND,PROT}}_{\text{LOSSY,RNDTAG}}(0)$ and $\mathsf{ProtRewind}^{\text{ZRND}}_{\text{LOSSY,RNDTAG}}(0)$. We conclude that

Lemma 18. $\mathsf{SimRewind}^{ZRND,PROT}_{LOSSY,RNDTAG}(0) \approx_p \mathsf{ProtRewind}^{ZRND}_{LOSSY,RNDTAG}(0)$

This concludes the proof of Theorem 8.

L Secure Coinflips Based on MPC

In this section we show how to implement $\mathcal{F}_{\text{RA+MPC+CF}}$ based on $\mathcal{F}_{\text{RA+MPC}}$. Thus, we need to describe a protocol that runs assuming $\mathcal{F}_{\text{RA+MPC}}$ is available. The only new thing we need to implement is the coin flip, and this is done by asking $\mathcal{F}_{\text{RA+MPC}}$ to evaluate an appropriate function and give the output to a number of committees. Concretely, the function $f_{\text{RSS}}(P_1, \ldots, P_n)$ creates a standard robust secret sharing of a random value modulo N, playing the role of the coin. The members of the committee $P_1, \ldots, P_n$ each learn a share and can then later reveal the value of the coin by sending shares to all players. Here, $P_1, \ldots, P_n$ should be understood as roles, that map to n random actual parties under the random permutation defined by $\mathcal{F}_{\text{RA+MPC}}$. Note that technically the function just gives the outputs to n consecutive parties. The permutation of the output unto roles is done by $\mathcal{F}_{\text{RA+MPC}}$.

We first describe the function to compute:

1. $f_{\text{RSS}}(P_1, \ldots, P_n)$ chooses random values $coin, a_1, \ldots, a_t \in \mathbb{Z}_N^*$, defines a polynomial

$$p(X) = \mathsf{coin} + a_1 X + \cdots + a_t X^t \bmod N$$

   and sets $\mathsf{sh}_j = p(j),\ j = 1, \ldots, n$.
2. For $i = 1, \ldots, n$, chooses $\alpha_i \in \mathbb{Z}_N^*$ at random. For $i, j = 1, \ldots, n$, choose $\beta_{i,j} \in \mathbb{Z}_N^*$ at random. Set $\mathsf{mac}_{i,j} = \alpha_i \mathsf{sh}_j + \beta_{i,j} \bmod N$.
3. Define the output out_j, for $i = j, \ldots, n$ as follows:

$$\mathsf{out}_j = \mathsf{sh}_j, \alpha_j, \beta_{j,1}, \ldots, \beta_{j,n}, \mathsf{mac}_{1,j}, \ldots, \mathsf{mac}_{n,j},$$

   and output out_j to P_j.

The idea is that sh_j is the actual share of coin, the α_j and $\beta_{j,i}$ are keys that can verify authentication codes for other shares, and the $\mathsf{mac}_{i,j}$-values are authentication codes for sh_j that can be verified using key material from other players. It is well known that these macs are information theoretically secure: given $\mathsf{sh}_j, \mathsf{mac}_{i,j}$, producing a different pair $\mathsf{sh}'_j, \mathsf{mac}'_{i,j}$ satisfying $\alpha_i \mathsf{sh}'_j + \beta_{i,j} = \mathsf{mac}'_{i,j}$ requires that you guess α_i, which happens with negligible probability $1/N$.

In the protocol below, players will receive a subset of the out_j's, some of which may be incorrect. Given a received value sh_j, we will say that sh_j *has support from* P_i if it is the case that $\mathsf{mac}_{i,j} = \alpha_i \mathsf{sh}_j + \beta_{i,j}$, where $\alpha_i, \beta_{i,j}$ are the values received from P_i.

The protocol to implement $\mathcal{F}_{\text{RA+MPC+CF}}$ in Figure 38 is very simple and is only described for a single coin, the extension to several coins is trivial.

Protocol Coinflip.

1. The protocol connects the interface of $\mathcal{F}_{\text{RA+MPC+CF}}$ directly to the corresponding interface of $\mathcal{F}_{\text{RA+MPC}}$, except for the part relating to coinflip.
2. If a coinflip ordered in the current batch, ask $\mathcal{F}_{\text{RA+MPC}}$ to compute $f_{\text{RSS}}(P_1, \ldots, P_n)$, where $P_1, \ldots, P_n$ stand for the next n available roles. It uses two batches on $\mathcal{F}_{\text{RA+MPC}}$ for each batch of $\mathcal{F}_{\text{RA+MPC+CF}}$. The first batch is used to emulate the batch of role assignment and MPC in one batch of $\mathcal{F}_{\text{RA+MPC+CF}}$. The second batch is used to generate coin-flips.
3. The Flip Coin command is implemented by having P_j send out_j to all players.
4. Each player waits until it has received $n-t$ shares that have support from at least $n-t$ players. Interpolate a value *coin* from the first $t+1$ such shares and output coin.

Fig. 38. Protocol for Coin-Flip

We first show that the protocol has liveness and outputs the correct value.

Lemma 19. *Except with negligible probability, each player in the* Coinflip *protocol eventually outputs the correct value of coin.*

Proof. It follows from the above security property of the macs that (with overwhelming probability) no incorrect share can have support from more than t players. Thus, a share having support from $n - t > t$ players can be assumed to be correct. On the other hand, a correct share will eventually have support from all $n-t$ honest players, since all $n-t$ honest messages are eventually delivered. So, a player can safely wait until it gets $n-t$ shares with enough support, and since these shares can be assumed correct, the output is correct. □

Theorem 9. *The* Coinflip *protocol implements $\mathcal{F}_{RA+MPC+CF}$ in the $\mathcal{F}_{RA+MPC}$-hybrid model.*

Proof. We describe a simulator for the protocol. When a coin is ordered, the simulator evaluates f_{RSS} but sets $\mathsf{coin} = 0$. It hands the resulting out_j-values to P_j for corrupt P_j and stores the honest values. Until the coin is to be revealed, if a new player P_j is corrupted, hand out_j to P_j. When coin is leaked from $\mathcal{F}_{\text{RA+MPC+CF}}$, interpolate a polynomial $g(X)$, such that $g(0) = \mathsf{coin}$ and $g(j) = 0$ for all corrupt P_j. For each honest P_i, update out_i as

follows: set $\mathsf{sh}_i = \mathsf{sh}_i + g(i)$, and update all $\mathsf{mac}_{j,i}$ values such that the new value of sh_i and the new macs verify against all the key material. This is trivial by solving n linear equations. Send the resulting out_i values on behalf of the honest players. When enough messages are delivered to honest player P, send a Deliver command to $\mathcal{F}_{\mathrm{RA+MPC+CF}}$.

It is straightforward to verify that this simulation is statistically indistinguishable from the protocol. The data from honest players and resource functionality seen by the environment have exactly the same distribution as in the real protocol, so by (the proof of) Lemma 19, the only source of error is when a corrupt player successfully forges a share, which happens with negligible probability. □