Guaranteed Output in $O(\sqrt{n})$ Rounds for Round-Robin Sampling Protocols*

Ran Cohen
cohenran@idc.ac.il
Reichman University

Jack Doerner
j@ckdoerner.net
Northeastern University

Yashvanth Kondi
ykondi@ccs.neu.edu
Northeastern University

abhi shelat
abhi@neu.edu
Northeastern University

September 22, 2022

Abstract

We introduce a notion of *round-robin* secure sampling that captures several protocols in the literature, such as the "powers-of-tau" setup protocol for pairing-based polynomial commitments and zk-SNARKs, and certain verifiable mixnets.

Due to their round-robin structure, protocols of this class inherently require n sequential broadcast rounds, where n is the number of participants.

We describe how to compile them generically into protocols that require only $O(\sqrt{n})$ broadcast rounds. Our compiled protocols guarantee output delivery against *any* dishonest majority. This stands in contrast to prior techniques, which require $\Omega(n)$ sequential broadcasts in most cases (and sometimes many more). Our compiled protocols permit a certain amount of adversarial bias in the output, as all sampling protocols with guaranteed output must, due to Cleve's impossibility result (STOC'86). We show that in the context of the aforementioned applications, this bias is harmless.

*A preliminary version [CDKs22] of this work appeared in *EUROCRYPT 2022*.

Contents

1 Introduction

In many settings it is desirable for a secure multiparty computation (MPC) protocol to *guarantee output delivery*, meaning that regardless of the actions taken by an adversary, all honest parties always learn their outputs from the computation. This property is needed, for example, in any use of secure computation that creates a critical *public output*, such as securely sampling the setup parameters needed for a blockchain system, etc. The recently-introduced powers-of-tau protocol [BGM17, GKM+18, KMSV21] does precisely this, and achieves guaranteed output delivery against $n-1$ corruptions, though it allows the adversary to bias the output. Motivated by this construction, this work presents a foundational study of sampling protocols that share the same abstract core properties as the powers-of-tau protocol. Concretely, we ask what resources are required for such protocols to guarantee output delivery, and in particular, how many rounds are required? We begin by describing existing methods for guaranteeing output delivery in MPC.

The seminal result of Cleve [Cle86] showed that unless a majority of parties are assumed to be honest, certain functions cannot be computed even with *fairness* (meaning that if the adversary learns the output then so do all honest parties). In the two-party setting, a series of works culminated with a full characterization of all finite-domain Boolean functions that can be computed with guaranteed output delivery [GHKL08, ALR13, Mak14, Ash14, ABMO15]. Our understanding is limited in the multiparty setting: only a handful of functions are known to be securely computable with guaranteed output delivery (e.g., the Boolean-OR and majority functions) [GK09, CL17, Dac20]. In fact, for $n > 3$, only Boolean OR is known to achieve guaranteed output delivery against $n-1$ corruptions without bias.

The Boolean-OR protocol of Gordon and Katz [GK09] inherently requires a *linear* number of broadcast rounds relative to the party count. It extends the folklore "player-elimination technique" (originally used in the honest-majority setting [GMW87, Gol04]) to the dishonest-majority case by utilizing specific properties of the Boolean-OR function. In a nutshell, the n parties iteratively run a related secure computation protocol with *identifiable abort* [IOZ14, CL17], meaning that if the protocol aborts without output, it is possible to identify at least one dishonest party. Since the abort may be conditioned on learning the putative output, this paradigm only works if the putative output is simulatable, which is the case for Boolean OR. If the protocol aborts, the dishonest party is identified and expelled, and the remaining parties restart the computation with a default input for the cheater (0 in case of Boolean OR). Because $n-1$ dishonest parties can force this process to repeat $n-1$ times, the overall round complexity must be $\Omega(n)$.[1]

[1] Surprisingly, if a *constant fraction* of the parties are assumed to be honest, this linear round complexity can be reduced to any super-constant function; e.g., $O(\log^* n)$ [CHOR22].

The $1/p$ relaxation. A closer look at Cleve's attack [Cle86] reveals that *any* r-round coin-tossing protocol that completes with a common output bit is exposed to an inverse-polynomial bias of $\Omega(1/r)$; it is a natural line of inquiry to attempt to achieve as tight a bias in the output as possible. Unfortunately, as far as we know, this approach creates a dependence of the round complexity on the number of parties that is typically *much worse* than linear. The first r-round, $(n-1)$-secure coin-tossing protocols assumed only one-way functions, but had a relatively large bias of $O(n/\sqrt{r})$ [ABC+85, Cle86].[2] Optimal bias of $O(1/r)$ was achieved for two parties [MNS16] and for three parties (up to polylog factors) [HT17], assuming oblivious transfer (OT). The state of the art for coin-tossing is the work of Buchbinder et al. [BHLT17] where the bias is $\tilde{O}\big(n^3 \cdot 2^n/r^{0.5+1/(2^{n-1}-2)}\big)$, which improves upon prior works [ABC+85, Cle86] for $n = O(\log\log r)$, i.e., when the number of rounds is *doubly exponential* in n (e.g., for a constant number of parties the bias translates to $O(1/r^{1/2+\Theta(1)})$).[3]

Towards generalizing the coin-tossing results, Gordon and Katz [GK12] relaxed the standard MPC security definition to capture bias via $1/p$-*secure computation*, where the protocol is secure with all but inverse-polynomial probability, as opposed to all but negligible.[4] They showed feasibility for any randomized two-party functionality with a polynomial-sized range and impossibility for certain functionalities with super-polynomial-sized domains and ranges. Beimel et al. [BLOO11] extended $1/p$-secure computation to the multiparty setting and presented protocols realizing certain functionalities with polynomial-sized ranges. However, their protocols again have round counts *doubly exponential* in n and only support a constant number of parties. Specifically, if the size of the range of a function is $g(\lambda)$, then the round complexity for computing that function with $1/p$-security is $(p(\lambda) \cdot g(\lambda))^{2^{O(n)}}$.

In sum, the $1/p$-relaxation requires many more rounds and is limited to functionalities with a polynomial-sized range. Many useful tasks, such as the sampling of cryptographic keys (which must be drawn from a range of super-polynomial size) cannot be achieved via this technique.

Biased-Sampling of Cryptographic Keys. Fortunately, some applications of MPC that require guaranteed output delivery can indeed tolerate quite large bias. A long line of works in the literature consider the problem of random sampling of *cryptographic objects* in which each party contributes its *own* public share in such a way that combining the public shares yields the public output, but even the joint view of $n-1$ secret shares remains useless. Protocols that follow this pattern give a rushing adversary the ability to see the public con-

[2] For a *constant fraction* of honest parties, the bias was improved to $O(1/\sqrt{r-n})$ [BOO15] and later to $O(1/\sqrt{r-\log^* n})$ [CHOR22].

[3] Optimal bias can be achieved for a constant number of parties if $t < 2n/3$ parties are corrupted [BOO15] (or if $t < 3n/4$, up to polylog factors [AO16]). Again, those protocols require a *doubly exponential* round complexity in n.

[4] Formally, there exists a polynomial p such that every attack on the "real-world" execution of the protocol can be simulated in the "ideal-world" computation such that the output of both computations cannot be distinguished in polynomial-time with more than $1/p(\lambda)$ probability.

tribution of the honest parties first, and only later choose the secrets of the corrupted parties. This approach permits the adversary to inflict a statistically large bias on the distribution of the public output (for example, forcing the output to always end in 0). However, the effect of this bias on the corresponding secret is hidden from the adversary due to the hardness of the underlying cryptographic primitive.

For some simple cryptographic objects (e.g., collectively sampling $x \cdot G$ [5]), there are *single-round* sampling protocols, known as *Non-Interactive Distributed Key Generation (NIDKG)* schemes [Sta96, FS01]. Interestingly, a classic construction for (interactive) distributed key generation by Pedersen [Ped91] in the honest-majority setting was found by Gennaro et al. [GJKR99] to unintentionally permit adversarial bias, which the same authors later proved can be tolerated in a number of applications [GJKR03].

For more complex cryptographic objects, the contributions of the parties cannot come in parallel. A few protocols are known in which the parties must each contribute only once, but they must contribute sequentially. We refer to these as *round-robin* protocols. Among them are the "powers-of-tau" protocol [BGM17, GKM+18, KMSV21] and variants of Abe's verifiable mixnets [Abe99, BKRS18], about which we will have more to say below. The round-robin approach inherently requires $\Omega(n)$ broadcast rounds.

For some cryptographic objects, the state-of-the-art sampling protocols do not guarantee output, but achieve security with identifiable abort. Multiparty RSA modulus generation [BF01, HMR+19, FLOP18, CCD+20, CHI+21] is a key example. Applying the player-elimination technique in this setting gives the adversary *rejection-sampling* capabilities, since the adversary can repeatedly learn the outcome of an iteration of the original protocol and then decide whether to reject it by actively cheating with a party (who is identified and eliminated), or accept it by playing honestly. An adversary that controls $n-1$ parties can reject $n-1$ candidate outputs before it must accept one. This *may* be different than inducing a plain bias, since the adversary can affect the distribution of the honest parties' contributions, but in this work we show that for certain tasks the two are the same. Regardless, the broadcast-round complexity of this approach is, again, inherently $\Omega(n)$.

To summarize, with the exception of NIDKG protocols a few specific tasks, all known techniques in the study of guaranteed output delivery with bias inherently require $\Omega(n)$ broadcast rounds (and sometimes even $\Omega(2^{2^n})$). It was our initial intuition that $\Omega(n)$ rounds were a barrier. Our main result is overcoming this intuitive barrier for an interesting class of functionalities.

1.1 Our Contributions

Our main contribution is to develop a new technique for constructing secure computation protocols that guarantee output delivery with bias using $O(\sqrt{n})$

[5] Where G is a generator of a group of order q written in additive notation, and x is a shared secret from $\mathbb{Z}_q$.

broadcast rounds while tolerating an arbitrary number of corruptions. Prior state-of-the-art protocols for the same tasks require n broadcast rounds. Moreover, our work stands in contrast to the folklore belief that realizing such functionalities with guaranteed output delivery *inherently* requires $\Omega(n)$ rounds.

Our technique applies to the sampling of certain cryptographic objects for which there exist *round-robin* sampling protocols, with a few additional properties. This class is nontrivial: it includes both the powers-of-tau and verifiable-mixnet constructions mentioned previously. The combination of scalability in n with security against $n-1$ corruptions is particularly important as it allows for better distribution of trust (given that there need only be a single honest party) than is possible with $\Omega(n)$-round protocols. Indeed, well-known real-world ceremonies for constructing the powers-of-tau-based setup parameters for zk-SNARK protocols involved just a few participants [BCG+15] and later one hundred participants [BGM17]. Our aim is to develop methods that allow thousands to millions of participants to engage in such protocols, which naturally requires a sublinear round complexity.

Though our techniques are model-agnostic, we formulate all of our results in the UC model. Specifically, we construct a *compiler* for round-robin protocols, and formally incorporate the adversary's bias into our ideal functionalities, as opposed to achieving only $1/p$-security [GK12].

The Basic Idea. The transformation underlying our compiler uses the "player-simulation technique" that goes back to Bracha [Bra87] and is widely used in the Byzantine agreement and MPC literature (e.g., [HM00, IPS08]) as well as the "player-elimination framework" [GMW87, Gol04]. We partition the set of n players into $\sqrt{n}$ subsets of size $\sqrt{n}$ each, and then construct a protocol that proceeds in at most $O(\sqrt{n})$ phases, with $O(1)$ rounds per phase. The key invariant of our technique is that in each phase, either one subset is able to *make progress* towards an output (and are thus able to halt), or if no subset succeeds, then at least one player from each active subset can be identified as cheating and removed from the next phase.

Applying our technique requires two key properties of the original protocol which we group under the moniker "strongly player-replaceable round-robin." We do not know precisely what kinds of functions can be computed by such protocols, but the literature already contains several examples. This issue is not new, as prior works in the literature must also resort to describing function classes by the "presence of an embedded XOR" [GHKL08] or the "size of domain or range" [BLOO11]. In our case, the restriction is defined by the existence of an *algorithm* with certain properties that can be used to compute the function.

Motivating Protocol: Powers of Tau. Before we give a more detailed explanation of our technique, it will be useful to recall a simplified version of the *powers-of-tau* protocol of Bowe, Gabizon, and Miers [BGM17]. Throughout, we assume synchronous communication, and a malicious adversary that can statically corrupt an arbitrary subset of the parties. The powers-of-tau protocol

was designed for generating setup parameters for Groth's zk-SNARK [Gro16]. Given an elliptic-curve group $\mathbb{G}$ generated by the point G, our simplified version will output $\{\tau \cdot G, \tau^2 \cdot G, \ldots, \tau^d \cdot G\}$, where d is public and τ is secret.

The protocol's invariant is to maintain as an intermediate result a vector of the same form as the output. In each round, the previous round's vector is rerandomized by a different party. For example, if the intermediate result of the first round is a vector $\{\tau_1 \cdot G, \tau_1^2 \cdot G, \ldots, \tau_1^d \cdot G\}$, then in round two the second party samples τ_2 uniformly and broadcasts $\{\tau_1 \cdot \tau_2 \cdot G, \tau_1^2 \cdot \tau_2^2 \cdot G, \ldots, \tau_1^d \cdot \tau_2^d \cdot G\}$, which it can compute by exponentiating each element of the previous vector. It also broadcasts a zero-knowledge proof that it knows the discrete logarithm of each element with respect to the corresponding element of the previous vector, and that the elements are related in the correct way.

It is not hard to see that a malicious party can bias the output, as Cleve's impossibility requires, and variants of this protocol have attempted to reduce the bias by forcing parties to speak twice [BCG+15, BGG18], using "random beacons" as an external source of entropy [BGM17], or considering restricted forms of *algebraic adversaries* [FKL18, KMSV21] in the random-oracle model.

Round-Robin Sampling Protocols. The powers-of-tau protocol has a simple structure shared by other (seemingly unrelated) protocols [Abe99, BKRS18], which we now attempt to abstract. First, observe that it proceeds in a round-robin fashion, where in every round a single party speaks over a broadcast channel, and the order in which the parties speak can be arbitrary. Furthermore, the message that each party sends depends only on public information (such as the transcript of the protocol so far, or public setup such as a common random string) and freshly-tossed private random coins known only to the sending party. The next-message function does not depend on private-coin setup such as a PKI, or on previously-tossed coins. *Strongly player-replaceable round-robin protocols*—the kind supported by our compiler—share these properties.

Next, we generalize this protocol-structure to arbitrary domains. We denote the "public-values" domain by $\mathbb{V}$ (corresponding to $\mathbb{G}^d$ in our simplified example) and the "secret-values" domain by $\mathbb{W}$ (corresponding to $\mathbb{Z}_q$). Consider an *update function* $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ (corresponding to the second party's "rerandomization" function, *sans proofs*) and denote by $\pi_{\mathsf{RRSample}}(f, n, u)$ the corresponding n-party round-robin protocol for some common public input value $u \in \mathbb{V}$ (corresponding to, e.g., $\{G, \ldots, G\}$). In addition to the basic powers-of-tau protocol and its variants [BGM17, GKM+18, KMSV21], this abstraction captures an additional interesting protocol from the literature: verifiable mixnets [BKRS18], where the parties hold a vector of ciphertexts and need to sample a random permutation.

Generalizing to Pre-transformation Functionality. Having defined the class of protocols, we specify a corresponding ideal functionality that these protocols realize in order to apply our compiler. This "pre-transformation functionality" is rather simple and captures the inherent bias that can be induced

by the adversary. Specifically, the functionality starts with the common public input u, and then samples a uniform secret value $w \in \mathbb{W}$ and updates u with w to yield a new public (intermediate) value $v := f(u, w)$. The functionality shows v to the adversary, and allows the adversary free choice of a bias value $x \in \mathbb{W}$ with which it updates v to yield the final output $y := f(v, x)$. For the specific case of powers-of-tau, this corresponds to an honest party picking a secret τ_1 and broadcasting $\{\tau_1 \cdot G, \tau_1^2 \cdot G, \ldots, \tau_1^d \cdot G\}$, and then the *adversary* choosing τ_2 (conditioned on the honest party's output) and broadcasting $\{\tau_1 \cdot \tau_2 \cdot G, \tau_1^2 \cdot \tau_2^2 \cdot G, \ldots, \tau_1^d \cdot \tau_2^d \cdot G\}$.

For update function $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ and common public input $u \in \mathbb{V}$, we denote by $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$ the n-party variant of the pre-transformation functionality. Proving that the round-robin protocol realizes this functionality boils down to realizing the a zero-knowledge proof that f has been correctly applied. We prove the following theorem:

Theorem 1.1 (Pre-Transformation Security, Informal)**.** *Let $n \in \mathbb{N}$, let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be an update function, and let $u \in \mathbb{V}$. Under these conditions, $\pi_{\mathsf{RRSample}}(f, n, u)$ realizes $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$ in the $\mathcal{F}_{\mathsf{NIZK}}$-hybrid model within n broadcast rounds.*

Theorem 1.1 gives the first *modular* analysis in the simulation paradigm of (a version of) the powers-of-tau protocol; this is opposed to other security analyses (e.g., [BGM17, KMSV21]) that give a monolithic security proof and explicitly avoid simulation-based techniques. On one hand, the modular approach allows the use of the powers-of-tau protocol to generate setup for other compatible constructions that otherwise rely on a trusted party, such as polynomial commitments [KZG10]. On the other hand, different instantiations of $\mathcal{F}_{\mathsf{NIZK}}$ give different security guarantees for the protocol: a universally composable (UC) NIZK in the CRS model yields a corresponding UC-secure protocol, a random-oracle-based NIZK yields security in the random-oracle model, and a knowledge-of-exponent-based NIZK yields stand-alone, non-black-box security in the plain model.

Round-Reducing Compiler. Let us now return to our main conceptual contribution: a compiler that reduces the round complexity of the round-robin protocols described above from n broadcast rounds to $O(\sqrt{n})$.

Let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be an update function and $u \in \mathbb{V}$ a common public input as before, and let $m < n$ be integers (without loss of generality, consider n to be an exact multiple of m). Given an m-party protocol $\pi_{\mathsf{RRSample}}(f, m, u)$ executed in m rounds by parties $\mathcal{Q}_1, \ldots, \mathcal{Q}_m$ (who speak sequentially), let $\mathbf{g}_j$ be the next-message function of $\mathcal{Q}_j$. The compiled protocol $\pi_{\mathsf{Compiler}}(\pi_{\mathsf{RRSample}}(f, m, u), n, u, m)$ will be executed by n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$.

The compiled protocol will organize its parties into m committees, and each committee will execute an (n/m)-party MPC protocol in order to *jointly* evaluate the next-message functions of parties in the original protocol. For ease of exposition, we will say that each committee in this new protocol acts as a

virtual party in the original, which proceeds in virtual rounds. The MPC protocol must be secure with *identifiable abort* [IOZ14, CL17] against any number of corruptions; that is, either all honest parties obtain their outputs or they all identify at least one cheating party.

Furthermore, the MPC must provide *public verifiability* [BDO14, SV15] in the sense that every party that is *not* in a particular committee must also learn that committee's output (or the identities of cheating parties), and be assured that the output is well-formed (i.e., compatible with the transcript, for some set of coins) even if the entire committee is corrupted. This is similar to the notions of *publicly identifiable abort* [KZZ16] and *restricted identifiable abort* [CHOR22].

In the i^{th} round, *all* of the committees will attempt to emulate the party $\mathcal{Q}_i$ of the original protocol, in parallel. If a party is identified as a cheater at any point, it is excluded from the rest of the computation. At the conclusion of all MPC protocols for the first round, one of two things must occur: either all committees aborted, in which case at least m cheating parties are excluded, and each committee re-executes the MPC protocol with the remaining parties, or else at least one committee completed with an output. In the latter case, let j be the minimal committee-index from those that generated output, and denote the output of committee j by $\mathbf{a}_i$. Next, all committees (except for committee j, which disbands) proceed as if the virtual party $\mathcal{Q}_i$ had broadcasted $\mathbf{a}_i$ in the i^{th} round, and continue in a similar way to emulate party $\mathcal{Q}_{i+1}$ in round $i+1$. Note that at a certain point all remaining committees may be fully corrupted, and cease sending messages. This corresponds to the remaining virtual parties being corrupted and mute in the virtual protocol; in this case all of the remaining committee members are identified as cheaters. The compiled protocol proceeds in this way until the virtualized copy of $\pi_{\mathsf{RRSample}}(f, m, u)$ is complete.

If the generic MPC protocol that underlies each virtual party requires constant rounds, then the entire protocol completes in $O(m + n/m)$ rounds, and if we set $m = \sqrt{n}$, we achieve a round complexity of $O(\sqrt{n})$, as desired. So long as there is at least one honest party, one virtual party is guaranteed to produce an output at some point during this time, which means that the compiled protocol has the same output delivery guarantee as the original.

Post-Transformation Functionality. Although the compiled protocol $\pi_{\mathsf{Compiler}}(\pi_{\mathsf{RRSample}}(f, m, u), n, u, m)$ emulates the original $\pi_{\mathsf{RRSample}}(f, m, u)$ in some sense, it does not necessarily realize $\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)$ as the original protocol does, because the adversary has additional rejection-sampling capabilities that allow for additional bias. We therefore specify a second ideal functionality $\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)$, where r is a bound on the number of rejections the adversary is permitted; setting this bound to 0 coincides with $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$.

As in $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$, the functionality begins by sampling $w \leftarrow \mathbb{W}$, computing $v = f(u, w)$ and sending v to the adversary, who can either accept or reject. If the adversary accepts then it returns $x \in \mathbb{W}$ and the functionality outputs $y = f(v, x)$ to everyone; if the adversary rejects, then the functionality samples another $w \leftarrow \mathbb{W}$, computes $v = f(u, w)$, and sends v to the adver-

sary, who can again either accept or reject. The functionality and the adversary proceed like this for up to r iterations, or until the adversary accepts some value.

Theorem 1.2 (Post-Transformation Security, Informal)**.** *Let $m < n$ be integers and let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ and $u \in \mathbb{V}$ be as above. Assume that $\pi_{\mathsf{RRSample}}(f, m, u)$ realizes $\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)$ using a suitable NIZK protocol within m broadcast rounds, and that the next-message functions of $\pi_{\mathsf{RRSample}}(f, m, u)$ can be securely computed with identifiable abort and public verifiability in a constant-number of rounds. Let $r = m + \lceil n/m \rceil$. Under these conditions, $\pi_{\mathsf{Compiler}}(\pi_{\mathsf{RRSample}}(f, m, u), n, u, m)$ realizes $\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)$ within $O(r)$ broadcast rounds.*

Although $\pi_{\mathsf{Compiler}}(\pi_{\mathsf{RRSample}}(f, m, u), n, u, m)$ does not necessarily realize $\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)$ for every f, we show that it somewhat-unexpectedly does if the update function f satisfies certain properties. Furthermore, we show that these properties are met in the cases of powers-of-tau and mixnets.

Theorem 1.3 (Equivalence of Pre- and Post-Transformation Security, Informal)**.** *Let $n, r \in \mathbb{N}$, let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be a* homomorphic *update function, and let $u \in \mathbb{V}$ be a common public input. If a protocol π realizes $\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)$ then π also realizes $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$.*

Powers of Tau and Polynomial Commitments. A polynomial-commitment scheme enables one to commit to a polynomial of some bounded degree d, and later open evaluations of the polynomial. The pairing-based scheme of Kate et al. [KZG10] requires trusted setup of the form $\{G, \tau \cdot G, \tau^2 \cdot G, \ldots, \tau^d \cdot G\} \in \mathbb{G}^{d+1}$, for some elliptic-curve group $\mathbb{G}$. The security of the scheme reduces to the d-strong Diffie-Hellman assumption (d-SDH) [BB04]. We show that if the setup is not sampled by a trusted party, but instead computed (with bias) by our protocol (either the round-robin or compiled variation), there is essentially no security loss.

Theorem 1.4 (Generating Setup for SDH, Informal)**.** *If there exists a PPT adversary that can break a d-SDH challenge generated by an instance of our protocol in which it has corrupted $n-1$ parties, then there exists a PPT adversary that can win the standard (unbiased) d-SDH game with the same probability.*

zk-SNARKs with Updateable Setup. Several recent zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) have featured *updatable* trusted setup, and have security proofs that hold so long as at least one honest party has participated in the update process [GKM+18, MBKM19, GWC19, CHM+20]. Since their proofs already account for adversarial bias and the form of their trusted setup derives from the setup of Kate et al. [KZG10], our protocols can be employed for an asymptotic improvement upon the best previously known update procedure.

Verifiable Mixnets. A verifiable mixnet is a multiparty protocol by which a group of parties can shuffle a set of encrypted inputs, with the guarantee that no corrupt subset of the parties can learn the permutation that was applied or prevent the output from being delivered, and the property that non-participating observers can be convinced that the shuffle was computed correctly. Prior constructions, such as the work of Boyle et al. [BKRS18], involve random shuffling and re-encryption in a round-robin fashion, and their security proofs already consider bias of exactly the sort our protocol permits. Thus, it is natural to apply our compiler, yielding the first verifiable mixnet that requires sublinear broadcast rounds.

Concrete Efficiency. While our primary goal in this work is optimizing round complexity, a round-efficient protocol is not useful in practice if it has unfeasibly high (but polynomially bounded) communication or computation complexity. As evidence of the practicality of our technique, we relax our round complexity goal to $O(\sqrt{n}\log d)$, and give an additional, non-generic construction that specifically computes the powers-of-tau, along with an analysis of its concrete costs.

1.2 Open Questions

Our work initiates the foundational study of "rejection-sampling protocols" with guaranteed output delivery, and uncovers various open questions that we leave for future research. For example, we do not know whether we can achieve sublinear broadcast complexity for sampling functionalities that are not known to be realized by round-robin protocols, or whether we can reduce the broadcast complexity of any non-trivial sampling functionality other than distributed key generation to $o(\sqrt{n})$ under standard assumptions. Presenting a lower bound seems like a challenging task, since indistinguishability obfuscation (iO) *may* suffice for achieving a *single-round* protocol, based on techniques for non-interactive multiparty key agreement [BZ14, KRS15]. However, even this is unclear to us.

2 Preliminaries

Notation. We use $=$ for equality, $:=$ for assignment, $\leftarrow$ for sampling from a distribution, $\equiv$ for distributional equivalence, $\approx_c$ for computational indistinguishability, and $\approx_s$ for statistical indistinguishability. In general, single-letter variables are set in *italic* font, function names are set in sans-serif font, and string literals are set in slab-serif font. We use $\mathbb{V}$, $\mathbb{W}$, $\mathbb{X}$, and $\mathbb{Y}$ for unspecified domains, but we use $\mathbb{G}$ for a group, $\mathbb{F}$ for a field, $\mathbb{Z}$ for the integers, $\mathbb{N}$ for the natural numbers, and Σ_d for the permutations over d elements. We use λ to denote the computational security parameter.

Vectors and arrays are given in bold and indexed by subscripts; thus $\mathbf{a}_i$ is the i^{th} element of the vector $\mathbf{a}$, which is distinct from the scalar variable a. When we wish to select a row or column from a multi-dimensional array, we place a $*$

in the dimension along which we are not selecting. Thus $\mathbf{b}_{*,j}$ is the j^{th} column of matrix $\mathbf{b}$, $\mathbf{b}_{j,*}$ is the j^{th} row, and $\mathbf{b}_{*,*} = \mathbf{b}$ refers to the entire matrix. We use bracket notation to generate inclusive ranges, so $[n]$ denotes the integers from 1 to n and $[5,7] = \{5,6,7\}$. On rare occasions, we may use one vector to index another: if $\mathbf{a} \coloneqq [2,7]$ and $\mathbf{b} \coloneqq \{1,3,4\}$, then $\mathbf{a}_\mathbf{b} = \{2,4,5\}$. We use $|x|$ to denote the bit-length of x, and $|\mathbf{y}|$ to denote the number of elements in the vector $\mathbf{y}$. We use $\mathcal{P}_i$ to indicate an actively participating party with index i; in a typical context, there will be a fixed set of active participants denoted $\mathcal{P}_1, \ldots, \mathcal{P}_n$. A party that observes passively but remains silent is denoted $\mathcal{V}$.

For convenience, we define a function GenSID, which takes *any* number of arguments and deterministically derives a unique Session ID from them. For example $\mathsf{GenSID}(\mathsf{sid}, x, \mathsf{x})$ derives a Session ID from the variables sid and x, and the string literal "x."

Universal Composability, Synchrony, Broadcast, and Guaranteed Output Delivery. We consider a malicious PPT adversary who can statically corrupt any subset of parties in a protocol, and require all of our constructions to guarantee output delivery. Guaranteed output delivery is traditionally defined in the *stand-alone model* (e.g., [CL17]) and *cannot* be captured in the inherently asynchronous UC framework [Can01]. For concreteness, we will consider the synchronous UC modeling of Katz et al. [KMTZ13], which captures guaranteed termination in UC, but for clarity we will use standard UC notation. We note that our techniques do not rely on any specific properties of the model, and can be captured in any composable framework that supports synchrony, e.g., those of Liu-Zhang and Maurer [LM20] or Baum et al. [BDD+21].

In terms of communication, we consider all messages to be sent over an authenticated broadcast channel, sometimes denoted by $\mathcal{F}_{\mathsf{BC}}$, and do not consider any point-to-point communication. This is standard for robust MPC protocols in the dishonest-majority setting. Our protocols proceed in rounds, where all parties receive the messages sent in round $i-1$ before anyone sends a message for round i.

3 A Round-Reducing Compiler

The main result of our paper is a round-reducing compiler for round-robin sampling protocols. To be specific, our compiler requires three conditions on any protocol ρ that it takes as input: ρ must have a *broadcast-only round-robin structure*, it must be *strongly player-replaceable*, and it must UC-realize a specific functionality $\mathcal{F}_{\mathsf{PreTrans}}(f, \cdot, \cdot)$ for some function f. We define each of these conditions in turn, before describing the compiler itself in Section 3.1.

Definition 3.1 (Broadcast-Only Round-Robin Protocol)**.** *A protocol has a* broadcast-only round-robin structure *if the parties in the protocol send exactly one message each in a predetermined order, via an authenticated broadcast channel. We often refer to such protocols simply as* round-robin *protocols.*

Definition 3.2 (Strong Player-Replaceability)**.** *A protocol is* strongly player-replaceable *if no party has any secret inputs or keeps any secret state. That is, the next-message functions in a* strongly player-replaceable *protocol may take as input only public values and a random tape.*

Remark 3.3 (Strongly Player-Replaceable Round-Robin Protocols)**.** *If a protocol $\rho(n, u)$ for n parties with some common input $u \in \mathbb{V}$ conforms to Definitions 3.1 and 3.2, then it can be represented as a vector of functions $\mathbf{g}_1, \ldots, \mathbf{g}_{n+1}$ such that $\mathbf{g}_i$ for $i \in [n]$ is the next-message function of the i^{th} party. $\mathbf{g}_1$ takes $u \in \mathbb{V}$ and a vector of η uniform coins for some $\eta \in \mathbb{N}$ as input, and each succeeding function $\mathbf{g}_i$ for $i \in [2, n]$ takes u concatenated with the outputs of all previous functions in the sequence, plus η additional uniform coins. The last function, $\mathbf{g}_{m+1}$, does not take any coins, and can be run locally by anyone to extract the protocol's output from its transcript. We refer to protocols that meet these criteria as* SPRRR protocols *hereafter.*

Note that Definition 3.2 is somewhat more restrictive than the (non-strong) player-replaceability property defined by Chen and Micali [CM19]. Their definition forbids secret state but allows players to use some kinds of secret inputs (in particular, secret signature keys) in the next-message function, so long as every player is capable of computing the next message for any given round. We forbid such secret inputs, giving parties only an ideal authenticated broadcast channel by which to distinguish themselves from one another.

Finally, we define the biased sampling functionality that any input protocol ρ is required to realize. This functionality is parameterized by a function f which takes an input value from some space (denoted $\mathbb{V}$) and a randomization witness (from some space $\mathbb{W}$) and produces an output value (again in $\mathbb{V}$) deterministically. The functionality models sampling with adversarial bias by selecting a randomization witness w from $\mathbb{W}$ uniformly, rerandomizing the input value using w, and then providing the resulting intermediate v to the adversary, who can select a second (arbitrarily biased) randomization witness x from $\mathbb{W}$ to apply to v using f, in order to produce the functionality's output y. Note that the *only* requirement on f is that it has the same input and output domains, so that it can be applied repeatedly. It is not required to have any other properties (such as, for example, one-wayness).

Functionality 3.4. $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$**. Biased Sampling**

This functionality interacts with n actively participating parties denoted by $\mathcal{P}_1 \ldots \mathcal{P}_n$ and with the ideal adversary $\mathcal{S}$. It is also parameterized by an update function $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ and an arbitrary value $u \in \mathbb{V}$.

Sampling: On receiving $(\mathtt{sample}, \mathsf{sid})$ from at least one $\mathcal{P}_i$ for $i \in [n]$,

1. If a record of the form $(\mathtt{unbiased}, \mathsf{sid}, *)$ exists in memory, then ignore this message. Otherwise, continue with steps 2 and 3.
2. Sample $w \leftarrow \mathbb{W}$ and compute $v := f(u, w)$.

3. Store $(\texttt{unbiased}, \mathsf{sid}, v)$ in memory and send $(\texttt{unbiased}, \mathsf{sid}, v)$ to $\mathcal{S}$.

Bias: On receiving $(\texttt{proceed}, \mathsf{sid}, x)$ from $\mathcal{S}$, where $x \in \mathbb{W}$,

4. If the record $(\texttt{done}, \mathsf{sid})$ exists in memory, or if the record $(\texttt{unbiased}, \mathsf{sid}, v)$ does not exist in memory, then ignore this message. Otherwise, continue with steps 5 and 6.

5. Compute $y := f(v, x)$.

6. Store $(\texttt{done}, \mathsf{sid})$ in memory and send $(\texttt{output}, \mathsf{sid}, y)$ to all parties.

Note that this functionality never allows an abort or adversarially delayed output to occur, and thus it has guaranteed output delivery.[6] Now that all of the constraints on input protocols for our compiler are specified, and we can introduce a second functionality, which will be UC-realized by the compiled protocol, given a constraint-compliant input protocol. This second functionality is similar to $\mathcal{F}_{\mathsf{PreTrans}}$ and likewise has guaranteed output delivery, but it takes an additional parameter r, and allows the adversary to reject up to r potential honest randomizations before it supplies its bias and the output is delivered.

Functionality 3.5. $\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)$**. Rejection Sampling**

This functionality interacts with n actively participating parties denoted by $\mathcal{P}_1 \ldots \mathcal{P}_n$ and with the ideal adversary $\mathcal{S}$. It is also parameterized by an update function $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$, an arbitrary value $u \in \mathbb{V}$, and a rejection bound $r \in \mathbb{N}$.

Sampling: On receiving $(\texttt{sample}, \mathsf{sid})$ from at least one $\mathcal{P}_i$ for $i \in [n]$,

1. If a record of the form $(\texttt{candidate}, \mathsf{sid}, *, *)$ exists in memory, then ignore this message. Otherwise, continue with steps 2 and 3.

2. Sample $\mathbf{w}_1 \leftarrow \mathbb{W}$ and compute $\mathbf{v}_1 := f(u, \mathbf{w}_1)$.

3. Store $(\texttt{candidate}, \mathsf{sid}, 1, \mathbf{v}_1)$ in memory and send the same tuple to $\mathcal{S}$.

Rejection: On receiving $(\texttt{reject}, \mathsf{sid}, i)$ from $\mathcal{S}$, where $i \in \mathbb{N}$,

4. If $i > r$, or if either of the records $(\texttt{done}, \mathsf{sid})$ or $(\texttt{candidate}, \mathsf{sid}, i + 1, \mathbf{v}_{i+1})$ exists in memory, or if the record $(\texttt{candidate}, \mathsf{sid}, i, \mathbf{v}_i)$ does not exist in memory, then ignore this message. Otherwise, continue with steps 5 and 6.

5. Sample $\mathbf{w}_{i+1} \leftarrow \mathbb{W}$ and compute $\mathbf{v}_{i+1} := f(u, \mathbf{w}_{i+1})$.

[6] Formally, every party requests the output from the functionality, and the adversary can instruct the functionality to ignore a polynomially-bounded number of such requests [KMTZ13].

6. Store $(\texttt{candidate}, \mathsf{sid}, i+1, \mathbf{v}_{i+1})$ in memory and send the same tuple to $\mathcal{S}$.

Bias: On receiving $(\texttt{accept}, \mathsf{sid}, i, x)$ from $\mathcal{S}$, where $i \in \mathbb{N}$ and $x \in \mathbb{W}$,

7. If either of the records $(\texttt{done}, \mathsf{sid})$ or $(\texttt{candidate}, \mathsf{sid}, i+1, \mathbf{v}_{i+1})$ exists in memory, or if the record $(\texttt{candidate}, \mathsf{sid}, i, \mathbf{v}_i)$ does not exist in memory, then ignore the message. Otherwise, continue with steps 8 and 9.

8. Compute $y := f(\mathbf{v}_i, x)$.

9. Store $(\texttt{done}, \mathsf{sid})$ in memory and send $(\texttt{output}, \mathsf{sid}, y)$ to all parties.

Observe that for any f, any $n \in \mathbb{N}^+$, and any $u \in \mathbb{V}$, the functionality $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$ is equivalent to $\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, 0)$ (though some of the message names differ). In fact, there are some functions f under which we can prove that the ideal protocol involving $\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)$ UC-realizes $\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)$ for any $r \in \mathbb{N}$; we discuss this further in Section 4.

We note that instead of one function $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$, both $\mathcal{F}_{\mathsf{PreTrans}}$ and $\mathcal{F}_{\mathsf{PostTrans}}$ could easily have been parameterized by two different functions $f_1 : \mathbb{V} \times \mathbb{W} \to \mathbb{X}$ and $f_2 : \mathbb{X} \times \mathbb{Y} \to \mathbb{V}$, where f_1 is used for applying the honest randomization from $\mathbb{W}$, and f_2 for applying the adversarial bias from $\mathbb{Y}$. Under this change, all of the proofs of theorems in Section 3 go through exactly as written, with the appropriate substitutions. However, our protocols in Section 4 require $f_1 = f_2$, and we know of no other input protocol that UC-realizes $\mathcal{F}_{\mathsf{PreTrans}}$ for *distinct* values of f_1 and f_2, so we choose to present a simplified view with only one update function, f.

Finally, we must discuss the property of public verifiability. We model public verifiability as an abstract modifier for other functionalities. The parties interacting with any particular session of an unmodified functionality become the *active participants* in the modified functionality, but there may be additional parties, known as *observing verifiers*, who may register to receive outputs (potentially unbeknownst to the active participants) but do not influence the functionality in any other way. This corresponds to the protocol property whereby a protocol instance can be verified as having been run correctly by third parties who have access to only a transcript (obtained, for example, by monitoring broadcasts). Strongly player-replaceable broadcast-only protocols have this property naturally if they are secure against malicious corruptions: without any stored secrets, private communication, or interaction, the active parties have no additional verification power over anyone else.

Functionality 3.6. $[\![\mathcal{F}]\!]_{\mathsf{PV}}$. Public Verifiability for $\mathcal{F}$

The functionality $[\![\mathcal{F}]\!]_{\mathsf{PV}}$ is identical to the functionality $\mathcal{F}$, except that it interacts with an arbitrary number of additional *observing verification parties* (all of them denoted by $\mathcal{V}$, as distinct from the *actively participating parties* $\mathcal{P}_1$, $\mathcal{P}_2$, etc.). Furthermore, if *all* actively participating parties are

corrupt, then $[\![\mathcal{F}]\!]_{\mathsf{PV}}$ receives its random coins from the adversary $\mathcal{S}$.

Coin Retrieval: Whenever the code of $\mathcal{F}$ requires a random value to be sampled from the domain $\mathbb{X}$, then sample as $\mathcal{F}$ would if at least one of the active participants is honest. If all active participants are corrupt, then send $(\texttt{need-coin}, \mathsf{sid}, \mathbb{X})$ to $\mathcal{S}$, and upon receiving $(\texttt{coin}, \mathsf{sid}, x)$ such that $x \in \mathbb{X}$ in response, continue behaving as $\mathcal{F}$, using x as the required random value.

Observer Registration: Upon receiving $(\texttt{observe}, \mathsf{sid})$ from $\mathcal{V}$, remember the identity of $\mathcal{V}$, and if any message with the same sid is sent to *all* active participants in the future, then send it to $\mathcal{V}$ as well.

In the introduction, we have omitted discussion of public verifiability for the sake of simplicity and clarity, but in fact, all known input protocols for our compiler have this property (that is, they UC-realize $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, which is strictly stronger than $\mathcal{F}_{\mathsf{PreTrans}}$). Furthermore, we will show that given an input protocol that realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, the compiled protocol realizes $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$.

Note that when proving that a protocol realizes a functionality with public verifiability, we do not typically need to reason about security against malicious observing verifiers, since honest parties ignore any messages they send, and therefore there can be nothing in their view that the adversary cannot already obtain by monitoring the relevant broadcast channel directly.

3.1 The Compiler

We now turn our attention to the compiler itself. We direct the reader to Section 1.1 for an intuitive view of the compiler, via virtual parties and virtual rounds. With this intuitive transformation in mind, we now present a compiler which formalizes it and addresses the unmentioned corner cases. The compiler takes the form of a multiparty protocol $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ that is parameterized by a description of the original protocol ρ for m parties, and by the number of real, active participants n, the public input u for the original protocol, and the number of committees (i.e., virtual parties) m. Before describing π_{Compiler}, we must formalize the tool that each committee uses to emulate a virtual party. We do this via a UC functionality for generic MPC with identifiable abort.

Functionality 3.7. $\mathcal{F}_{\mathsf{SFE\text{-}IA}}(f, n)$**. SFE with Identifiable Abort [IOZ14]**

This functionality interacts with n actively participating parties denoted by $\mathcal{P}_1 \ldots \mathcal{P}_n$ and with the ideal adversary $\mathcal{S}$. It is also parameterized by a function, $f : \mathbb{X}_1 \times \ldots \times \mathbb{X}_n \to \mathbb{Y}$.

SFE: On receiving $(\texttt{compute}, \mathsf{sid}, \mathbf{x}_i)$ where $\mathbf{x}_i \in \mathbb{X}_i$ from every party $\mathcal{P}_i$ for $i \in [n]$,

1. Compute $y := f\left(\{\mathbf{x}_i\}_{i\in[n]}\right)$.
2. Send $(\texttt{candidate-output}, \mathsf{sid}, y)$ to $\mathcal{S}$, and receive $(\texttt{stooge}, \mathsf{sid}, c)$ in response.
3. If c is the index of a corrupt party, then send $(\texttt{abort}, \mathsf{sid}, c)$ to all parties. Otherwise, send $(\texttt{output}, \mathsf{sid}, y)$ to all parties.

In order to ensure that every party can identify the cheaters in committees that it is not a member of, we must apply $[\![\cdot]\!]_{\mathsf{PV}}$ to $\mathcal{F}_{\mathsf{SFE\text{-}IA}}$, which gives us *publicly verifiable identifiable abort*. We discuss a method for realizing this functionality in Section 3.2; see Lemma 3.15 for more details. We can now give a formal description of our compiler.

Protocol 3.8. $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$**. Round-reducing Compiler**

This compiler is parameterized by ρ, which is a strongly player-replaceable round-robin protocol with two parameters: the number of participants, which may be hardcoded as m, and a common public input value from the domain $\mathbb{V}$. Let $\mathbf{g}_1, \ldots, \mathbf{g}_{m+1}$ be the vector of functions corresponding to ρ as described in Remark 3.3, and let η be the number of coins that the first m functions require. The compiler is also parameterized by the party count $n \in \mathbb{N}^+$, the common public input $u \in \mathbb{V}$, and the committee count $m \in \mathbb{N}^+$ such that $m \leq n$. In addition to the actively participating parties $\mathcal{P}_\ell$ for $\ell \in [n]$, the protocol involves the ideal functionality $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$, and it may involve one or more observing verifiers, denoted by $\mathcal{V}$.

Sampling: Let $\mathbf{a}_0 := u$ and let $\mathbf{C}_{1,*,*}$ be a deterministic partitioning of $[n]$ into m balanced subsets. That is, for $i \in [m]$, let $\mathbf{C}_{1,i,*}$ be a vector indexing the parties in the i^{th} committee. Upon receiving $(\texttt{sample}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party repeats the following sequence of steps, starting with $k := 1$ and $\mathbf{j}_1 := 1$, incrementing k with each loop, and terminating the loop when $\mathbf{j}_k > m$

1. For all $i \in [m]$ (in parallel) each party $\mathcal{P}_\ell$ for $\ell \in \mathbf{C}_{k,i,*}$ samples $\boldsymbol{\omega}_\ell \leftarrow \{0,1\}^\eta$ and sends $(\texttt{compute}, \mathsf{GenSID}(\mathsf{sid}, k, i), \boldsymbol{\omega}_\ell)$ to $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$, where $\boldsymbol{\gamma}_{\mathbf{j}_k}$ is a function such that
$$\boldsymbol{\gamma}_{\mathbf{j}_k}\left(\{\boldsymbol{\omega}_\ell\}_{\ell\in\mathbf{C}_{k,i,*}}\right) \mapsto \mathbf{g}_{\mathbf{j}_k}\left(\mathbf{a}_{[0,\mathbf{j}_k]}, \bigoplus_{\ell\in\mathbf{C}_{k,i,*}} \boldsymbol{\omega}_\ell\right)$$
2. For all $i \in [m]$ (in parallel) each party $\mathcal{P}_\ell$ for $\ell \in [n] \setminus \mathbf{C}_{k,i,*}$ sends $(\texttt{observe}, \mathsf{GenSID}(\mathsf{sid}, k, i))$ to $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$ (thereby taking the role of verifier).

3. For all $i \in [m]$, all parties receive either $(\texttt{abort}, \mathsf{GenSID}(\mathsf{sid}, k, i), \mathbf{c}_{k,i})$ or $(\texttt{output}, \mathsf{GenSID}(\mathsf{sid}, k, i), \hat{\mathbf{a}}_{k,i})$ from $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\gamma_{\mathbf{j}_k}, |\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$. In the latter case, let $\mathbf{c}_{k,i} := \perp$.

4. If any outputs were produced in the previous step, then let ℓ be the smallest integer such that $(\texttt{output}, \mathsf{GenSID}(\mathsf{sid}, k, \ell), \hat{\mathbf{a}}_{k,\ell})$ was received. Let $\mathbf{j}_{k+1} := \mathbf{j}_k + 1$ and let $\mathbf{a}_{\mathbf{j}_k+1} := \hat{\mathbf{a}}_{k,\ell}$ and for every $i \in [m]$ let

$$\mathbf{C}_{k+1,i,*} := \begin{cases} \mathbf{C}_{k,i,*} \setminus \{\mathbf{c}_{k,i}\} & \text{if } i \neq \ell \\ \varnothing & \text{if } i = \ell \end{cases}$$

5. If no outputs were produced in Step 3, then let $\mathbf{j}_{k+1} := \mathbf{j}_k$ and for every $i \in [m]$ let

$$\mathbf{C}_{k+1,i,*} := \mathbf{C}_{k,i,*} \setminus \{\mathbf{c}_{k,i}\}$$

Finally, each party outputs $(\texttt{output}, \mathsf{sid}, \mathbf{g}_{m+1}(\mathbf{a}_m))$ to the environment when the loop terminates.

Verification: If there is an observing verifier $\mathcal{V}$, then upon receiving $(\texttt{observe}, \mathsf{sid})$ from the environment $\mathcal{Z}$, it repeats the following sequence of steps, starting with $k := 1$ and $\mathbf{j}_1 := 1$, incrementing k with each loop, and terminating the loop when $\mathbf{j}_k > m$.

6. $\mathcal{V}$ sends $(\texttt{observe}, \mathsf{GenSID}(\mathsf{sid}, k, i))$ to $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\gamma_{\mathbf{j}_k}, |\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$ for all $i \in [m]$, and receives either $(\texttt{abort}, \mathsf{GenSID}(\mathsf{sid}, k, i), \mathbf{c}_{k,i})$ or $(\texttt{output}, \mathsf{GenSID}(\mathsf{sid}, k, i), \hat{\mathbf{a}}_{k,i})$ in response.

7. $\mathcal{V}$ determines the value of $\mathbf{j}_{k+1}$ and $\mathbf{C}_{k+1,*,*}$ per the method in Steps 4 and 5.

Finally, $\mathcal{V}$ outputs $(\texttt{output}, \mathsf{sid}, \mathbf{g}_{m+1}(\mathbf{a}_m))$ to the environment when the loop terminates.

3.2 Proof of Security

In this section we provide security and efficiency proofs for our compiler. Our main security theorem (Theorem 3.9) is split into two sub-cases: the case that there is at least one honest active participant is addressed by Lemma 3.10, and the case that there are no honest active participants (but there is one or more honest observing verifiers) is addressed by Lemma 3.13. After this, we give a folklore method for realizing $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ in Lemma 3.15, and use it to prove our main efficiency result in Corollary 3.16.

Theorem 3.9. *Let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be an update function, let $u \in \mathbb{V}$, let $m \in \mathbb{N}^+$, and let ρ be an SPRRR protocol such that $\rho(m, u)$ UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ in the presence of a malicious adversary statically corrupting any number of actively participating parties. For every integer $n \geq m$, it*

holds that $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ *UC-realizes* $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, m + n/m)]\!]_{\mathsf{PV}}$ *in the presence of a malicious adversary statically corrupting any number of actively participating parties in the* $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$*-hybrid model.*

Proof. By conjunction of Lemmas 3.10 and 3.13. Since corruptions are static, a single simulator can be constructed that follows the code of either $\mathcal{S}_{\mathsf{Compiler}}$ (defined in the proof of Lemma 3.10) or $\mathcal{S}_{\mathsf{CompilerPV}}$ (defined in the proof of Lemma 3.13), depending on the number of active participants corrupted by the real-world adversary $\mathcal{A}$. □

Lemma 3.10. *Let* $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ *be an update function, let* $u \in \mathbb{V}$*, let* $m \in \mathbb{N}^+$*, and let* ρ *be an SPRRR protocol such that* $\rho(m, u)$ *UC-realizes* $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ *in the presence of a malicious adversary statically corrupting up to* $m - 1$ *actively participating parties. For every integer* $n \geq m$*, it holds that* $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ *UC-realizes* $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, m + n/m)]\!]_{\mathsf{PV}}$ *in the presence of a malicious adversary statically corrupting up to* $n - 1$ *actively participating parties in the* $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$*-hybrid model.*

Note that the above lemma also holds if the $[\![\cdot]\!]_{\mathsf{PV}}$ modifier is removed from *both* functionalities. This is straightforward to see, given the proof of the lemma as written, so we elide further detail. Regardless, because the proof of this lemma is our most interesting and subtle proof, upon which our other results rest, we will sketch it first, to give the reader an intuition, and *then* present the formal version afterward.

Proof Sketch. In this sketch we give an overview of the simulation strategy followed by the simulator $\mathcal{S}_{\mathsf{Compiler}}$ against a malicious adversary who corrupts up to $n - 1$ parties, using the same terminology and simplified, informal protocol description that we used to build an intuition about the compiler in Section 1.1. Recall that with the i^{th} protocol committee we associate an emulated "virtual" party $\mathcal{Q}_i$, for the purposes of exposition. We are guaranteed by the premise of Theorem 3.10, that there exists an ideal adversary $\mathcal{S}_{\rho,\mathcal{D}}$ that simulates a transcript of ρ for the dummy adversary $\mathcal{D}$ that corrupts up to $m - 1$ parties, while engaging in an ideal interaction with functionality $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ on $\mathcal{D}$'s behalf. The compiled protocol $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ represents a single instance of the original protocol ρ, but in each virtual round there is an m-way fork from which a single definitive outcome is selected (by the adversary) to form the basis of the next virtual round. The main idea behind $\mathcal{S}_{\mathsf{Compiler}}$ is that the forking tree can be pruned in each virtual round to include only the single path along which the a real honest party's contribution lies (or might lie, if no honest contribution has yet become a definitive outcome), and then $\mathcal{S}_{\rho,\mathcal{D}}$ can be used to translate between the protocol instances represented by these paths and the functionality $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, m, u, m)]\!]_{\mathsf{PV}}$.

At a high level, for each fresh candidate $\mathbf{v}_i$ produced by $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, m, u, m)]\!]_{\mathsf{PV}}$, the simulator $\mathcal{S}_{\mathsf{Compiler}}$ will invoke an instance of $\mathcal{S}_{\rho,\mathcal{D}}$, feed it all the (definitive-output) messages produced by the protocol thus far, and then feed it $\mathbf{v}_i$ in order to generate a corresponding honest-party

message that can be sent to the corrupted parties. It repeats this process until the adversary accepts the honest party's contribution in some virtual round κ, whereafter the last instance of $\mathcal{S}_{\rho,\mathcal{D}}$ (which was created in round κ) is fed the remaining protocol messages in order to extract the adversary's bias y. Let $h \in [n]$ index an honest party, and let θ index the committee in to which it belongs, (corresponding to $\mathcal{Q}_\theta$). The outline for $\mathcal{S}_{\mathsf{Compiler}}$ is as follows (dropping Session IDs for the sake of simplification):

1. Initialize $j := 1$, $k := 1$, $\mathbf{a}_0 := u$, $\kappa := \perp$.

2. Obtain a candidate $\mathbf{v}_k$ by sending either $\texttt{sample}$ (only when $k = 1$) or $(\texttt{reject}, k-1)$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$, and receiving $(\texttt{candidate}, k, \mathbf{v}_k)$ in response.

3. Invoke $\mathcal{S}_{\rho,\mathcal{D}}$ on protocol transcript $\mathbf{a}_*$ (each message being sent on behalf of a different corrupt party, and then send it $(\texttt{unbiased}, \mathbf{v}_k)$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$ in order to obtain the tentative protocol message $\hat{\mathbf{a}}_{k,\theta}$ of $\mathcal{Q}_\theta$.

4. Send $(\texttt{candidate-output}, \hat{\mathbf{a}}_{k,\theta})$ on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ to the corrupt parties in the committee indexed by θ, and wait for the adversary to either accept this output, or abort by blaming a corrupt committee-member.

5. Simultaneously, interact with the fully corrupt committees indexed by $[m]\backslash\{\theta\}$ on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ to learn the values of $\hat{\mathbf{a}}_{k,i}$ for $i \in [m]\backslash\{\theta\}$.

6. If any virtual parties produced non-aborting output during this virtual round, then let $i' \in [m]$ be the smallest number that indexes such a virtual party. Let $\mathbf{a}_j := \hat{\mathbf{a}}_{k,i'}$ (making the output of $\mathcal{Q}_{i'}$ definitive) and if $i' = \theta$ then set $\kappa := j$ and skip to Step 8; otherwise, increment j and k and return to Step 2, updating the committee partitioning to remove the committee corresponding to $\mathcal{Q}_{i'}$ (and to remove any cheating real parties from the other committees) as per the protocol.

7. If no virtual parties produced non-aborting output during this virtual round, then increment k (but *not* j), update the committee partitioning to remove the cheaters as per the protocol, and return to Step 2.

8. Once $\mathcal{Q}_\theta$ has produced a definitive output (in virtual round κ) and its underlying committee has disbanded, continue interacting with the other (fully corrupt) committees on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ until they have all either produced a definitive output (which is appended to $\mathbf{a}$) or become depleted of parties due to cheating. At this point, $\mathbf{a}_*$ should comprise a full transcript of protocol ρ. Some prefix of this transcript has already been transmitted to the final instance of $\mathcal{S}_{\rho,\mathcal{D}}$ (which was spawned in Step 2 during virtual round κ); send the remaining messages (those not in the prefix) to the last instance of $\mathcal{S}_{\rho,\mathcal{D}}$ as well, and it should output $(\texttt{proceed}, x)$ along with its interface to $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$. Send $(\texttt{accept}, \kappa, x)$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ and halt.

The only non-syntactic aspect in which the above simulation differs from the real protocol is as follows: whereas in the real protocol $\mathcal{Q}_\theta$ computes its message $\hat{\mathbf{a}}_{k,\theta}$ by running its honest code as per ρ (recall that this virtual party is realized by an invocation of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ by committee θ), in the simulation this value is produced by $\mathcal{S}_{\rho,\mathcal{D}}$ in consultation with $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]^{\mathsf{PV}}$. Observe, first, that the `reject` interface of $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]^{\mathsf{PV}}$ functions identically to an individual invocation of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ and second that the transcript produced by $\mathcal{S}_{\rho,\mathcal{D}}$ in its interaction with $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ is indistinguishable from a real execution of ρ. From these two observations, we can conclude that the above simulation is indistinguishable from a real execution of π_{Compiler} to any efficient adversary. □

With this simplified sketch of the proof completed, we now proceed to the full, formal proof of Lemma 3.10.

Proof of Lemma 3.10. By the premise, $\rho(m,u)$ realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f,m,u)]\!]^{\mathsf{PV}}$; that is,

$$\begin{aligned}&\forall \mathcal{A}\ \exists \mathcal{S}_{\rho,\mathcal{A}} \text{ s.t. } \forall \mathcal{Z},\\ &\quad \left\{\mathrm{REAL}_{\rho(m,u),\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\lambda,m\in\mathbb{N}^+,u\in\mathbb{V},z\in\{0,1\}^*}\\ &\approx_{\mathrm{c}} \left\{\mathrm{IDEAL}_{[\![\mathcal{F}_{\mathsf{PreTrans}}(f,m,u)]\!]^{\mathsf{PV}},\mathcal{S}_{\rho,\mathcal{A}}(m,u),\mathcal{Z}}(\lambda,z)\right\}_{\lambda,m\in\mathbb{N}^+,u\in\mathbb{V},z\in\{0,1\}^*}\end{aligned} \tag{1}$$

It follows from Equation 1 that there must exist an ideal adversary $\mathcal{S}_{\rho,\mathcal{D}}$ for the dummy adversary $\mathcal{D}$. In this proof, we construct a new ideal adversary, $\mathcal{S}_{\mathsf{Compiler}}$, which requires black-box access to $\mathcal{S}_{\rho,\mathcal{D}}$ and to a real-world adversary $\mathcal{A}$, and then prove that

$$\begin{aligned}&\forall \mathcal{A}\ \forall \mathcal{Z},\\ &\quad \left\{\mathrm{REAL}_{\pi_{\mathsf{Compiler}}(\rho,n,u,m),\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\substack{\lambda,n,m\in\mathbb{N}^+:n\geq m,\\ u\in\mathbb{V},z\in\{0,1\}^*}}\\ &\approx_{\mathrm{c}} \left\{\mathrm{IDEAL}_{[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,m+n/m)]\!]^{\mathsf{PV}},\mathcal{S}^{\mathcal{S}_{\rho,\mathcal{D}},\mathcal{A}}_{\mathsf{Compiler}}(\rho,f,n,u,m),\mathcal{Z}}(\lambda,z)\right\}_{\substack{\lambda,n,m\in\mathbb{N}^+:n\geq m,\\ u\in\mathbb{V},z\in\{0,1\}^*}}\end{aligned} \tag{2}$$

We begin by specifying $\mathcal{S}_{\mathsf{Compiler}}$, after which our proof of Equation 2 proceeds via a sequence of hybrid experiments of length $m + n/m + 2$.

Simulator 3.11. $\mathcal{S}^{\mathcal{S}_{\rho,\mathcal{D}},\mathcal{A}}_{\mathsf{Compiler}}(\rho,f,n,u,m)$**. Against Dishonest Majority**

This simulator is parameterized by a player-replaceable round-robin protocol ρ for m participants, and by the update function $f : \mathbb{V}\times\mathbb{W}\to\mathbb{V}$ such that $\rho(m,u)$ UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f,m,u)]\!]^{\mathsf{PV}}$ in the presence of a malicious adversary corrupting up to $m-1$ actively participating parties. This simulator has black-box access to the simulator $\mathcal{S}_{\rho,\mathcal{D}}$ for the dummy adversary $\mathcal{D}$, and to the adversary $\mathcal{A}$ who is guaranteed to corrupt no more than $n-1$ active participants. n is the number of actively participating parties in the protocol to be simulated, $m\in[n]$ is the number of committees, and

$u \in \mathbb{V}$ is a common public input.

Init: On initial activation for the session ID sid, $\mathcal{S}_{\mathsf{Compiler}}$ begins emulating in its head an instance of the real-world experiment for $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ for the adversary $\mathcal{A}$ (to which $\mathcal{S}_{\mathsf{Compiler}}$ has black-box access). Let this emulated experiment be referred to as SUBEXPT_0, and let the values of $\boldsymbol{\gamma}_*$, $\mathbf{a}_*$, $\hat{\mathbf{a}}_{*,*}$, and $\mathbf{C}_{*,*,*}$ henceforth be defined relative to their values in this sub-experiment. Furthermore, $\mathcal{S}_{\mathsf{Compiler}}$ plays the role of the ideal oracle $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ in SUBEXPT_0. $\mathcal{S}_{\mathsf{Compiler}}$ forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$ in SUBEXPT_0, and vice versa, and when $\mathcal{A}$ announces that it wishes to corrupt a set of parties indexed by $\mathbf{P}^* \subset [n]$, $\mathcal{S}_{\mathsf{Compiler}}$ corrupts the corresponding parties in its own experiment. Upon learning $\mathbf{P}^*$, $\mathcal{S}_{\mathsf{Compiler}}$ arbitrarily chooses a single honest party index $h \in [n] \setminus \mathbf{P}^*$. Let θ be the index of the committee that contains $\mathcal{P}_h$.

Sampling:

- Upon receiving $(\texttt{compute}, \mathsf{GenSID}(\mathsf{sid}, k, i), \boldsymbol{\omega}_p)$ on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$ from a corrupt party $\mathcal{P}_p$ for $p \in \mathbf{C}_{k,i,*} \cap \mathbf{P}^*$ in SUBEXPT_0, if a record of the form $(\texttt{sample-accepted}, \mathsf{sid}, *)$ does not exist in memory:

  1. If $k = 1$, then on behalf of $\mathcal{P}_p$, $\mathcal{S}_{\mathsf{Compiler}}$ sends $(\texttt{sample}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)]\!]_{\mathsf{PV}}$ if no such message has already been sent on behalf of $\mathcal{P}_p$.
  2. If $\mathbf{C}_{k,i,*} \cap \mathbf{P}^* = \mathbf{C}_{k,i,*}$ (that is, all active parties associated with this instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ are corrupt), then $\mathcal{S}_{\mathsf{Compiler}}$ follows the code of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$.
  3. If $\mathbf{C}_{k,i,*} \cap \mathbf{P}^* \neq \mathbf{C}_{k,i,*}$ but $i \neq \theta$ (that is, $\mathcal{P}_h$ is not associated with this instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$, but at least one *other* honest party is), and a message of the form $(\texttt{compute}, \mathsf{GenSID}(\mathsf{sid}, k, i), *)$ has previously been received from every party $\mathcal{P}_{p'}$ for $p' \in (\mathbf{C}_{k,i,*} \cap \mathbf{P}^*) \setminus \{p\}$, then $\mathcal{S}_{\mathsf{Compiler}}$ waits (asynchronously) for the record $(\texttt{candidate}, \mathsf{sid}, k, \mathbf{v}_k)$ to appear in its memory. When this happens, $\mathcal{S}_{\mathsf{Compiler}}$ emulates $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ by following its code, and also emulates the honest parties indexed by $\mathbf{C}_{k,i,*} \setminus \mathbf{P}^*$ in their interaction with $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ by following their code.
  4. If $\mathbf{C}_{k,i,*} \cap \mathbf{P}^* \neq \mathbf{C}_{k,i,*}$ and $i = \theta$ (that is, $\mathcal{P}_h$ is associated with this instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$), and a message of the form $(\texttt{compute}, \mathsf{GenSID}(\mathsf{sid}, k, i), *)$ has previously been received from every party $\mathcal{P}_{p'}$ for $p' \in (\mathbf{C}_{k,i,*} \cap \mathbf{P}^*) \setminus \{p\}$, then $\mathcal{S}_{\mathsf{Compiler}}$ waits (asynchronously) for the record $(\texttt{candidate}, \mathsf{sid}, k, \mathbf{v}_k)$ to appear in its memory. When this happens, $\mathcal{S}_{\mathsf{Compiler}}$ begins emulating a new instance of the ideal-world experiment for $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ with ideal adversary $\mathcal{S}_{\rho,\mathcal{D}}$ (to which $\mathcal{S}_{\mathsf{Compiler}}$ has black-box access) and m

parties, in which it plays the role of the environment, the ideal oracle $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, and all honest parties. Let this new emulated experiment be referred to as SUBEXPT_k. In its role as the environment and using the interface of $\mathcal{D}$, $\mathcal{S}_{\mathsf{Compiler}}$ corrupts the parties indexed by $[m] \setminus \{\mathbf{j}_k\}$ and sequentially instructs each $\mathcal{P}_{p'}$ for $p' \in [\mathbf{j}_{k-1}]$ to broadcast $\mathbf{a}_{p'}$. Then, in its role as $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, $\mathcal{S}_{\mathsf{Compiler}}$ waits for a message $(\mathtt{sample}, \mathsf{sid})$ to be sent by $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of each $\mathcal{P}_{p'}$ for $p' \in [\mathbf{j}_{k-1}]$, whereupon it sends $(\mathtt{unbiased}, \mathsf{sid}, \mathbf{v}_k)$ to $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$ and waits for a response.

- On receiving $(\mathtt{candidate}, \mathsf{sid}, k, \mathbf{v}_k)$ from $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)]\!]_{\mathsf{PV}}$, $\mathcal{S}_{\mathsf{Compiler}}$ stores this message in memory.

- Upon receiving a message $\hat{\mathbf{a}}_k$ via the interface of $\mathcal{D}$ from $\mathcal{S}_{\rho,\mathcal{D}}$ (in its role representing the honest party) in SUBEXPT_k, $\mathcal{S}_{\mathsf{Compiler}}$ finds the value of i such that $h \in \mathbf{C}_{k,i,*}$ in SUBEXPT_0, and then sends $(\mathtt{candidate\text{-}output}, \mathsf{GenSID}(\mathsf{sid}, k, i), \hat{\mathbf{a}}_k)$ to $\mathcal{A}$ on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$. When $\mathcal{A}$ responds with $(\mathtt{stooge}, \mathsf{GenSID}(\mathsf{sid}, k, i), c)$, if $c \in \mathbf{C}_{k,i,*} \cap \mathbf{P}^*$, then $\mathcal{S}_{\mathsf{Compiler}}$ sends $(\mathtt{abort}, \mathsf{GenSID}(\mathsf{sid}, k, i), c)$ to the corrupt parties on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ in SUBEXPT_0. If $c \notin \mathbf{C}_{k,i,*} \cap \mathbf{P}^*$, then $\mathcal{S}_{\mathsf{Compiler}}$ sends $(\mathtt{output}, \mathsf{GenSID}(\mathsf{sid}, k, i), \hat{\mathbf{a}}_k)$ to the corrupt parties on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ in SUBEXPT_0.

- When the value of $\mathbf{a}_k$ becomes finalized in SUBEXPT_0, if no record of the form $(\mathtt{sample\text{-}accepted}, \mathsf{sid}, *)$ exists in memory, and $\mathbf{a}_k$ was delivered as the output of an instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ with session ID $\mathsf{GenSID}(\mathsf{sid}, k, i)$ for i such that $h \notin \mathbf{C}_{k,i,*}$, then $\mathcal{S}_{\mathsf{Compiler}}$ sends $(\mathtt{reject}, \mathsf{sid}, k)$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$. Otherwise, if no record of the form $(\mathtt{sample\text{-}accepted}, \mathsf{sid}, *)$ exists in memory, then $\mathcal{S}_{\mathsf{Compiler}}$ stores $(\mathtt{sample\text{-}accepted}, \mathsf{sid}, k)$ in memory.

Bias:

- Upon receiving $(\mathtt{compute}, \mathsf{GenSID}(\mathsf{sid}, k, i), \boldsymbol{\omega}_p)$ on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$ from a corrupt party $\mathcal{P}_p$ for $p \in \mathbf{C}_{k,i,*} \cap \mathbf{P}^*$ in SUBEXPT_0, if a record of the form $(\mathtt{sample\text{-}accepted}, \mathsf{sid}, *)$ exists in memory:

  1. If $i = \theta$ (that is, $\mathcal{P}_h$ is associated with this instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$), then $\mathcal{S}_{\mathsf{Compiler}}$ does nothing.
  2. If $i \neq \theta$ (that is, $\mathcal{P}_h$ is not associated with this instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$), then $\mathcal{S}_{\mathsf{Compiler}}$ follows the code of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ and, if necessary, any *other* honest parties that interact with $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$.

- When the value of $\mathbf{a}_k$ becomes finalized in SUBEXPT_0, if a record of the form $(\mathtt{sample\text{-}accepted}, \mathsf{sid}, \kappa)$ exists in memory, then in SUBEXPT_κ,

$\mathcal{S}_{\mathsf{Compiler}}$ instructs $\mathcal{S}_{\rho,\mathcal{D}}$ via the interface of $\mathcal{D}$ to broadcast $\mathbf{a}_k$ on behalf of the corrupt $\mathcal{P}_k$.

- Upon receiving $(\texttt{proceed}, \mathsf{sid}, x)$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$ from $\mathcal{S}_{\rho,\mathcal{D}}$ in SUBEXPT_κ, if the record $(\texttt{sample-accepted}, \mathsf{sid}, \kappa)$ does not exist in memory, then $\mathcal{S}_{\mathsf{Compiler}}$ does nothing. Otherwise, $\mathcal{S}_{\mathsf{Compiler}}$ sends $(\texttt{accept}, \mathsf{sid}, \kappa, x)$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)]\!]_{\mathsf{PV}}$.

Our sequence of hybrid experiments begins with the real-world experiment, as specified in Equation 2. Specifically,

$$\mathcal{H}_0 := \left\{\text{REAL}_{\pi_{\mathsf{Compiler}}(\rho,n,u,m),\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\lambda,n,m\in\mathbb{N}^+:n\geq m;u\in\mathbb{V},z\in\{0,1\}^*}$$

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$, except that $\mathcal{Z}$ now communicates with a single, monolithic entity, $\mathcal{S}$, which internally emulates an instance of the real-world experiment for $\mathcal{A}$ (to which $\mathcal{S}$ has black-box access), in which $\mathcal{S}$ itself plays the roles of all parties and oracles (excluding $\mathcal{Z}$ and $\mathcal{A}$), following their code exactly as specified in $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$, and forwarding all messages between the emulated experiment's environment and $\mathcal{Z}$. Let this emulated experiment be denoted by SUBEXPT_0; henceforth in this sequence of hybrid experiments, all variables defined in the protocol $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ are defined with respect to the instance of $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ emulated by $\mathcal{S}$. When $\mathcal{S}$ learns from $\mathcal{A}$ the value of $\mathbf{P}^*$, $\mathcal{S}$ arbitrarily chooses $h \in [n] \setminus \mathbf{P}^*$ (this set is always nonempty, per the premise) and sets θ to be the index of the committee that contains $\mathcal{P}_h$, but it does not (yet) use these variables. Because these changes are purely syntactical, $\mathcal{H}_0 = \mathcal{H}_1$.

Hybrid $\mathcal{H}_{k+1}$ $\forall k \in [m + n/m]$. Hybrid $\mathcal{H}_{k+1}$ is identical to $\mathcal{H}_k$, except that $\mathcal{S}$ now has black-box access to $\mathcal{S}_{\rho,\mathcal{D}}$ (if it didn't previously), and *if* there is an activation of the functionality $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta}, *|)]\!]_{\mathsf{PV}}$ in SUBEXPT_0 with session ID $\mathsf{GenSID}(\mathsf{sid}, k, \theta)$, then instead of following the code of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$, $\mathcal{S}$ begins emulating a new instance of the ideal-world experiment for $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ to $\mathcal{S}_{\rho,\mathcal{D}}$ (to which $\mathcal{S}$ has black-box access). Let this new emulated experiment be referred to as SUBEXPT_k. In SUBEXPT_k, $\mathcal{S}$ plays the role of the environment, the ideal oracle $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, and all honest parties.

In its role as the environment for SUBEXPT_k, and using the interface of $\mathcal{D}$, $\mathcal{S}$ corrupts parties indexed by $[m] \setminus \{\mathbf{j}_k\}$ and sequentially instructs each $\mathcal{P}_p$ for $p \in [\mathbf{j}_{k-1}]$ to broadcast $\mathbf{a}_p$. Then, in its role as $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, $\mathcal{S}$ waits for a message $(\texttt{sample}, \mathsf{sid})$ to be sent by $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of each $\mathcal{P}_p$ for $p \in [\mathbf{j}_{k-1}]$, whereupon it follows the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$ to produce a response for $\mathcal{S}_{\rho,\mathcal{D}}$. $\mathcal{S}_{\rho,\mathcal{D}}$ then replies with $\hat{\mathbf{a}}_{k,\theta}$ via the interface of $\mathcal{D}$ (in its role representing the honest party), and $\mathcal{S}$ then sends $(\texttt{candidate-output}, \mathsf{GenSID}(\mathsf{sid}, k, \theta), \hat{\mathbf{a}}_{k,\theta})$ to $\mathcal{A}$ in SUBEXPT_0 on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$. When $\mathcal{A}$ responds with $(\texttt{stooge}, \mathsf{GenSID}(\mathsf{sid}, k, \theta), c)$, if $c \in \mathbf{C}_{k,\theta,*} \cap \mathbf{P}^*$, then $\mathcal{S}$ sends $(\texttt{abort}, \mathsf{GenSID}(\mathsf{sid}, k, \theta), c)$ to the corrupt parties on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ in SUBEXPT_0. If $c \notin \mathbf{C}_{k,\theta,*} \cap \mathbf{P}^*$, then $\mathcal{S}$ sends

$(\texttt{output}, \mathsf{GenSID}(\mathsf{sid}, k, \theta), \hat{\mathbf{a}}_{k,\theta})$ to the corrupt parties on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]^{\mathsf{PV}}$ in SUBEXPT_0.

When the value of $\mathbf{a}_k$ becomes finalized in SUBEXPT_0, if $\mathbf{a}_k$ was delivered as the output of an instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta,*}|)]\!]^{\mathsf{PV}}$ with session ID $\mathsf{GenSID}(\mathsf{sid}, k, \theta)$ (i.e., $\mathbf{a}_k = \hat{\mathbf{a}}_{k,\theta}$), then for the remainder of SUBEXPT_0, whenever a value $\mathbf{a}_p$ for $p \in [\mathbf{j}_k + 1, m]$ becomes finalized, $\mathcal{S}$ uses the interface of $\mathcal{D}$ to instruct $\mathcal{P}_p$ to broadcast $\mathbf{a}_p$ in SUBEXPT_k. After SUBEXPT_0 completes, $\mathcal{S}_{\rho,\mathcal{D}}$ should send a message $(\texttt{proceed}, \mathsf{sid}, x)$ to $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ in SUBEXPT_k, whereupon $\mathcal{S}$ follows the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ to produce an output y, which it sends to $\mathcal{Z}$ as the output of the honest parties.

We will now show that if there exists a pair $(\mathcal{A}, \mathcal{Z})$ such that $\mathcal{Z}$ can distinguish $\mathcal{H}_{k+1}$ from $\mathcal{H}_k$ with advantage ϵ, then we can construct a new environment $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$ that uses $(\mathcal{A}, \mathcal{Z})$ in a black-box way and has the same advantage ϵ in breaking the UC-Security of ρ (that is, in distinguishing the real-world experiment for ρ from the ideal-world experiment for $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$).

Observe that in $\mathcal{H}_{k+1}$, $\mathcal{S}$ internally uses calls to $\mathcal{S}_{\rho,\mathcal{D}}$ and the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ in order to emulate the instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta,*}|)]\!]^{\mathsf{PV}}$ with session ID $\mathsf{GenSID}(\mathsf{sid}, k, \theta)$. On the other hand, in $\mathcal{H}_k$, $\mathcal{S}$ follows the code of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta,*}|)]\!]^{\mathsf{PV}}$, and by implication generates the functionality's output via evaluation of $\boldsymbol{\gamma}_{\mathbf{j}_k}$, which is the honest party's next-message function in ρ. Thus we construct our reduction:

Algorithm 3.12. $\mathcal{Z}^{\mathcal{Z},\mathcal{A}}_{\mathsf{Reduction}\text{-}k}$**. Distinguisher for** ρ **and** $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$

This functionality expects to interact with the dummy adversary $\mathcal{D}$ in the experiment given by Equation 1. It has additional black-box access to $\mathcal{Z}, \mathcal{A}$, a distinguishing pair for the experiment given by Equation 2.

On initial activation, $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$ begins emulating an experiment to $\mathcal{Z}$ and $\mathcal{A}$ (to which it has black-box access). This experiment is identical to $\mathcal{H}_{k+1}$, except where it involves the functionality $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta,*}|)]\!]^{\mathsf{PV}}$ with session ID $\mathsf{GenSID}(\mathsf{sid}, k, \theta)$ in SUBEXPT_0: specifically, where $\mathcal{S}$ in $\mathcal{H}_{k+1}$ would interact with $\mathcal{S}_{\rho,\mathcal{D}}$ via the $\mathcal{D}$ interface in order to compute values to output on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta,*}|)]\!]^{\mathsf{PV}}$, $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$ instead performs the same interaction with the actual $\mathcal{D}$ (which has the same interface) in its own, non-emulated experiment.

Notice that if $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$ finds itself in an instance of the real-world experiment for ρ, then the value output by $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta,*}|)]\!]^{\mathsf{PV}}$ with session ID $\mathsf{GenSID}(\mathsf{sid}, k, \theta)$ in $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$'s emulated SUBEXPT_0 will be computed by an actual honest party running its next-message function $\mathbf{g}_{\mathbf{j}_k}$; consequently, the view of $\mathcal{Z}$ in the SUBEXPT_0 emulated by $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$ is distributed identically to its view in the SUBEXPT_0 emulated by $\mathcal{S}$ in $\mathcal{H}_k$ in this case.

Notice furthermore if $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$ finds itself in an instance of the ideal-world experiment for $\mathcal{F}_{\mathsf{PreTrans}}$, then the value output by $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,\theta,*}|)]\!]^{\mathsf{PV}}$ with $\mathsf{GenSID}(\mathsf{sid}, k, \theta)$ in $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$'s emulated SUBEXPT_0 will be computed by an

instance of $\mathcal{S}_{\rho,\mathcal{D}}$ that interacts with $\mathcal{F}_{\mathsf{PreTrans}}$; consequently, the view of $\mathcal{Z}$ in the SUBEXPT_0 emulated by $\mathcal{Z}_{\mathsf{Reduction}\text{-}k}$ is distributed identically to its view in the SUBEXPT_0 emulated by $\mathcal{S}$ in $\mathcal{H}_{k+1}$ in this case.

Thus, if there exists a pair $(\mathcal{A},\mathcal{Z})$ such that $\mathcal{Z}$ can distinguish $\mathcal{H}_{k+1}$ from $\mathcal{H}_k$ with advantage ϵ, then with advantage ϵ we can also distinguish

$$\left\{\text{REAL}_{\rho(n,u),\mathcal{D},\mathcal{Z}^{\mathcal{Z},\mathcal{A}}_{\mathsf{Reduction}\text{-}k}}(\lambda,z)\right\}_{\lambda,n\in\mathbb{N}^+,u\in\mathbb{V},z\in\{0,1\}^*}$$

from

$$\left\{\text{IDEAL}_{[\![\mathcal{F}_{\mathsf{PreTrans}}(f,n,u)]\!]_{\mathsf{PV}},\mathcal{S}_{\rho,\mathcal{D}}(n,u),\mathcal{Z}^{\mathcal{Z},\mathcal{A}}_{\mathsf{Reduction}\text{-}k}}(\lambda,z)\right\}_{\lambda,n\in\mathbb{N}^+,u\in\mathbb{V},z\in\{0,1\}^*}$$

for every ideal-adversary $\mathcal{S}_{\rho,\mathcal{D}}$, and by Equation 1 (the premise of our theorem) it follows that $\mathcal{H}_{k+1}\approx_{\text{c}}\mathcal{H}_k$.

Hybrid $\mathcal{H}_{m+n/m+2}$. This hybrid is identical to $\mathcal{H}_{m+n/m+1}$, except for the following four changes:

1. $\mathcal{S}$ no longer communicates with $\mathcal{Z}$ on behalf of the honest parties. Instead, we introduce the ideal oracle $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$, which communicates with $\mathcal{S}$ via the ideal-adversary's interface, and with $\mathcal{Z}$ via dummy honest parties.

2. In the context of SUBEXPT_1, when $\mathcal{S}$ receives a message $(\texttt{sample},\mathsf{sid})$ from $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, rather than following the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, $\mathcal{S}$ sends $(\texttt{sample},\mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ in its own experiment (if no such message has previously been sent), waits to receive $(\texttt{candidate},\mathsf{sid},1,\mathbf{v}_1)$ in response, and then sends $(\texttt{unbiased},\mathsf{sid},\mathbf{v}_1)$ to $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$.

3. Whenever in the context of SUBEXPT_k for some $k\in[2,m+n/m]$, $\mathcal{S}$ receives a message $(\texttt{sample},\mathsf{sid})$ from $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, rather than following the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, $\mathcal{S}$ sends $(\texttt{reject},\mathsf{sid},k-1)$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ in its own experiment (if no such message has previously been sent), waits to receive $(\texttt{candidate},\mathsf{sid},k,\mathbf{v}_k)$ in response, and then sends $(\texttt{unbiased},\mathsf{sid},\mathbf{v}_k)$ to $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$.

4. Whenever in the context of SUBEXPT_k for some $k\in[m+n/m]$, $\mathcal{S}$ receives a message $(\texttt{proceed},\mathsf{sid},x)$ from $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, *and* it is the case that $\mathbf{a}_k$ was delivered as the output of an instance of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k},|\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$ with session ID $\mathsf{GenSID}(\mathsf{sid},k,i)$ for i such that $h\notin\mathbf{C}_{k,i,*}$ in SUBEXPT_0, rather than following the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, $\mathcal{S}$ sends $(\texttt{accept},\mathsf{sid},k,x)$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ in its own experiment, and then halts.

Notice that the behavior of $\mathcal{S}$ in $\mathcal{H}_{m+n/m+2}$ is identical to the behavior of $\mathcal{S}_{\mathsf{Compiler}}$. Thus with the addition of the ideal oracle $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ and the dummy honest party, we have

$$\mathcal{H}_{m+\frac{n}{m}+2}=\left\{\text{IDEAL}_{[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,m+\frac{n}{m})]\!]_{\mathsf{PV}},\mathcal{S}^{\mathcal{S}_{\rho,\mathcal{D}},\mathcal{A}}_{\mathsf{Compiler}}(\rho,f,n,u,m),\mathcal{Z}}(\lambda,z)\right\}_{\substack{\lambda\in\mathbb{N}^+,\\ n,m\in\mathbb{N}^+:n\geq m,\\ u\in\mathbb{V},z\in\{0,1\}^*}}$$

and it remains to argue that $\mathcal{H}_{m+n/m+2}$ cannot be efficiently distinguished from $\mathcal{H}_{m+n/m+1}$. To this end, observe that in $\mathcal{H}_{m+n/m+1}$, $\mathcal{S}$ ran the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, and thus sampled a uniform $\mathbf{w}_k \leftarrow \mathbb{W}$ and then computed $\mathbf{v}_k := f(u, \mathbf{w}_k)$ for each $k \in [m + n/m]$ that had an associated experiment SUBEXPT_k. This is precisely the same calculation as is done by $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ in $\mathcal{H}_{m+n/m+2}$. Furthermore, the honest party's output to $\mathcal{Z}$ in $\mathcal{H}_{m+n/m+1}$ was calculated by $\mathcal{S}$ using the code of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$ as $y := f(\mathbf{v}_\kappa, x)$, where $\kappa \in [m + n/m]$ is the largest value for which there is an associated SUBEXPT_κ. This is precisely the same calculation as is done by $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ in $\mathcal{H}_{m+n/m+2}$. It follows that $\mathcal{H}_{m+n/m+2} = \mathcal{H}_{m+n/m+1}$.

This sequence of hybrids has consisted of two steps that are perfectly indistinguishable, and $m + n/m$ steps that are distinguishable with advantage no greater than ϵ, where ϵ is the maximum advantage of any environment in the game given by Equation 1. Thus we have that

$$\left\{\text{REAL}_{\pi_{\mathsf{Compiler}}(\rho,n,u,m),\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\lambda,n,m\in\mathbb{N}^+:n\geq m;u\in\mathbb{V},z\in\{0,1\}^*}$$

is distinguishable from

$$\left\{\text{IDEAL}_{[\![\mathcal{F}_{\mathsf{PostTrans}}(f_1,n,u,m+n/m)]\!]_{\mathsf{PV}},\mathcal{S}^{\mathcal{S}_{\rho,\mathcal{D}},\mathcal{A}}_{\mathsf{Compiler}}(\rho,f,n,u,m),\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda,n,m\in\mathbb{N}^+:n\geq m,\\ u\in\mathbb{V},z\in\{0,1\}^*}}$$

with probability no greater than $\epsilon \cdot (m + n/m)$, and Equation 2 holds. □

We now prove that the compiled protocol UC-realizes the functionality even if *all* active participants are corrupt.

Lemma 3.13. *Let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be an update function, let $u \in \mathbb{V}$, let $m \in \mathbb{N}^+$, and let ρ be an SPRRR protocol such that $\rho(m, u)$ UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ in the presence of an honest* observing verifier *and a malicious adversary statically corrupting all m actively participating parties. For every integer $n \geq m$, it holds that $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ UC-realizes $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, m + n/m)]\!]_{\mathsf{PV}}$ in the presence of an honest observing verifier and a malicious adversary statically corrupting all n actively participating parties in the $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$-hybrid model.*

Proof. By the premise, Equation 1 holds. It follows that there must exist an ideal adversary $\mathcal{S}_{\rho,\mathcal{D}}$ for the dummy adversary $\mathcal{D}$. In this proof, we construct a new ideal adversary, $\mathcal{S}_{\mathsf{CompilerPV}}$, which requires black-box access to $\mathcal{S}_{\rho,\mathcal{D}}$ and to a real-world adversary $\mathcal{A}$ that corrupts all actively participating parties, and then prove that

$$\begin{aligned}&\forall\mathcal{A}\ \forall\mathcal{Z},\\ &\quad\left\{\text{REAL}_{\pi_{\mathsf{Compiler}}(\rho,n,u,m),\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda,n,m\in\mathbb{N}^+:n\geq m,\\ u\in\mathbb{V},z\in\{0,1\}^*}}\\ &\approx_{\text{c}}\left\{\text{IDEAL}_{[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,m+n/m)]\!]_{\mathsf{PV}},\mathcal{S}^{\mathcal{S}_{\rho,\mathcal{D}},\mathcal{A}}_{\mathsf{CompilerPV}}(\rho,f,n,u,m),\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda,n,m\in\mathbb{N}^+:n\geq m,\\ u\in\mathbb{V},z\in\{0,1\}^*}}\end{aligned} \tag{3}$$

We begin by specifying $\mathcal{S}_{\mathsf{Compiler}}$, after which our proof of Equation 3 proceeds via a sequence of hybrid experiments. Unlike in our proof of Theorem 3.10, all actively participating parties are corrupt in this case, and so there are no "honest contributions" for $\mathcal{S}_{\mathsf{CompilerPV}}$ to communicate to $\mathcal{A}$. Thus $\mathcal{A}$ can never do anything that corresponds to rejection, and $\mathcal{S}_{\mathsf{CompilerPV}}$ need not ever send a `reject` message to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$.

Simulator 3.14. $\mathcal{S}^{\mathcal{S}_{\rho,\mathcal{D}},\mathcal{A}}_{\mathsf{CompilerPV}}(\rho, f_1, f_2, n, u, m)$**. Against Full Corruption**

This simulator is parameterized by a player-replaceable round-robin protocol ρ for m participants, and by the update function $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ such that $\rho(m, u)$ UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ in the presence of a malicious adversary corrupting all actively participating parties. This simulator has black-box access to the simulator $\mathcal{S}_{\rho,\mathcal{D}}$ for the dummy adversary $\mathcal{D}$, and to the adversary $\mathcal{A}$ who is guaranteed to corrupt all active participants. n is the number of actively participating parties in the protocol to be simulated, $m \in [n]$ is the number of committees, and $u \in \mathbb{V}$ is a common public input.

Init: On initial activation for the session ID sid, $\mathcal{S}_{\mathsf{CompilerPV}}$ begins emulating in its head an instance of the real-world experiment for $\pi_{\mathsf{Compiler}}(\rho, n, u, m)$ for the adversary $\mathcal{A}$ (to which $\mathcal{S}_{\mathsf{CompilerPV}}$ has black-box access). Let this emulated experiment be referred to as SUBEXPT_0, and let the values of $\boldsymbol{\gamma}_*$ and $\mathbf{a}_*$ and $\mathbf{C}_{*,*,*}$ henceforth be defined relative to their values in this sub-experiment. Furthermore, $\mathcal{S}_{\mathsf{CompilerPV}}$ plays the role of the ideal oracle $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ in SUBEXPT_0. $\mathcal{S}_{\mathsf{CompilerPV}}$ forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$ in SUBEXPT_0, and vice versa. $\mathcal{S}_{\mathsf{CompilerPV}}$ corrupts all active participants.

Additionally, $\mathcal{S}_{\mathsf{CompilerPV}}$ begins emulating an instance of the ideal-world experiment for $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, m, u)]\!]_{\mathsf{PV}}$ with ideal adversary $\mathcal{S}_{\rho,\mathcal{D}}$ (to which $\mathcal{S}_{\mathsf{CompilerPV}}$ has black-box access) and m parties, in which it plays the role of the environment and the ideal oracle $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$. Let this new emulated experiment be referred to as SUBEXPT_1. In its role as the environment and using the interface of $\mathcal{D}$, $\mathcal{S}_{\mathsf{CompilerPV}}$ corrupts all active participants in SUBEXPT_1.

Sampling and Bias:

- Upon receiving $(\texttt{compute}, \mathsf{GenSID}(\mathsf{sid}, k, i), \boldsymbol{\omega}_p)$ on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\boldsymbol{\gamma}_{\mathbf{j}_k}, |\mathbf{C}_{k,i,*}|)]\!]_{\mathsf{PV}}$ from a corrupt party $\mathcal{P}_p$ for $p \in \mathbf{C}_{k,i,*} \cap \mathbf{P}^*$ in SUBEXPT_0, if $k = 1$, then on behalf of $\mathcal{P}_p$, $\mathcal{S}_{\mathsf{CompilerPV}}$ sends $(\texttt{sample}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)]\!]_{\mathsf{PV}}$ if no such message has already been sent on behalf of $\mathcal{P}_p$, and, regardless of the value of k, $\mathcal{S}_{\mathsf{CompilerPV}}$ follows the code of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$.
- When the value of $\mathbf{a}_k$ becomes finalized in SUBEXPT_0, $\mathcal{S}_{\mathsf{CompilerPV}}$ instructs

$\mathcal{S}_{\rho,\mathcal{D}}$ via the interface of $\mathcal{D}$ to broadcast $\mathbf{a}_k$ on behalf of the corrupt $\mathcal{P}_k$ in SUBEXPT_1.

- Upon receiving $(\texttt{need-coin}, \mathsf{sid}, \mathbb{W})$ from $[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,r)]\!]_{\mathsf{PV}}$, $\mathcal{S}_{\mathsf{CompilerPV}}$ forwards the request to $\mathcal{S}_{\rho,\mathcal{D}}$ in SUBEXPT_1 on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$. On receiving a reply from $\mathcal{S}_{\rho,\mathcal{D}}$, $\mathcal{S}_{\mathsf{CompilerPV}}$ forwards the reply to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$.
- On receiving $(\texttt{candidate}, \mathsf{sid}, 1, \mathbf{v}_1)$ from $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$, $\mathcal{S}_{\mathsf{CompilerPV}}$ sends $(\texttt{unbiased}, \mathsf{sid}, \mathbf{v}_1)$ to $\mathcal{S}_{\rho,\mathcal{D}}$ in SUBEXPT_1 on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$ and waits for a response.
- Upon receiving $(\texttt{proceed}, \mathsf{sid}, x)$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$ from $\mathcal{S}_{\rho,\mathcal{D}}$ in SUBEXPT_1, if the $\mathcal{S}_{\mathsf{CompilerPV}}$ sends $(\texttt{accept}, \mathsf{sid}, 1, x)$ to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$.

Our sequence of hybrid experiments begins with the real-world experiment, as specified in Equation 3. Specifically,

$$\mathcal{H}_0 := \left\{\text{REAL}_{\pi_{\mathsf{Compiler}}(\rho,n,u,m),\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\lambda,n,m\in\mathbb{N}^+:n\geq m;u\in\mathbb{V},z\in\{0,1\}^*}$$

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$, except that $\mathcal{Z}$ now communicates with a single, monolithic entity, $\mathcal{S}$, which internally emulates an instance of the real-world experiment for $\mathcal{A}$ (to which $\mathcal{S}$ has black-box access), in which $\mathcal{S}$ itself plays the roles of all oracles and the one or more honest verifying observers, following their code exactly as specified in $\pi_{\mathsf{Compiler}}(\rho,n,u,m)$, and forwarding all messages between the emulated experiment's environment and $\mathcal{Z}$. Let this emulated experiment be denoted by SUBEXPT_0; henceforth in this sequence of hybrid experiments, all variables defined in the protocol $\pi_{\mathsf{Compiler}}(\rho,n,u,m)$ are defined with respect to the instance of $\pi_{\mathsf{Compiler}}(\rho,n,u,m)$ emulated by $\mathcal{S}$. Because these changes are purely syntactical, $\mathcal{H}_0 = \mathcal{H}_1$.

Hybrid $\mathcal{H}_2$. Hybrid $\mathcal{H}_2$ is identical to $\mathcal{H}_1$, except that $\mathcal{S}$ now has black-box access to $\mathcal{S}_{\rho,\mathcal{D}}$, and its behavior changes in the following ways:

1. Upon initialization, $\mathcal{S}$ begins emulating a new instance of the ideal-world experiment for $[\![\mathcal{F}_{\mathsf{PreTrans}}(f,m,u)]\!]_{\mathsf{PV}}$ to $\mathcal{S}_{\rho,\mathcal{D}}$ (to which $\mathcal{S}$ has black-box access). Let this new emulated experiment be referred to as SUBEXPT_1. In SUBEXPT_1, $\mathcal{S}$ plays the role of the environment, the ideal oracle $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, and one or more honest observing verifiers. In its role as the environment for SUBEXPT_1, and using the interface of $\mathcal{D}$, $\mathcal{S}$ instructs $\mathcal{S}_{\rho,\mathcal{D}}$ to corrupt all actively participating parties, and then sequentially instructs each corrupt $\mathcal{P}_p$ for $p \in [m]$ to broadcast $\mathbf{a}_p$ whenever $\mathbf{a}_p$ becomes finalized in its own experiment.
2. Upon receiving an instruction $(\texttt{verify}, \mathsf{sid})$ from $\mathcal{Z}$ on behalf of some observing verifier $\mathcal{V}$, $\mathcal{S}$ does not follow the code of $\mathcal{V}$, but instead sends $(\texttt{need-coin}, \mathsf{sid}, \mathbb{W})$ to $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$. Upon receiving $(\texttt{coin}, \mathsf{sid}, w)$ in reply, $\mathcal{S}$ computes $v := f(u,w)$ and sends

$(\texttt{unbiased}, \mathsf{sid}, v)$ to $\mathcal{S}_{\rho,\mathcal{D}}$ on behalf of $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, and then, on receiving $(\texttt{proceed}, \mathsf{sid}, x)$ in reply, $\mathcal{S}$ computes $y := f(v, x)$ and sends $(\texttt{output}, \mathsf{sid}, y)$ to $\mathcal{Z}$ on behalf of $\mathcal{V}$. If $\mathcal{Z}$ activates further observing verifiers, they receive identical responses.

A reduction analogous to the one specified by Algorithm 3.12 can be used to invalidate Equation 1 with no loss in advantage given an $\mathcal{Z}$ and $\mathcal{A}$ that can distinguish $\mathcal{H}_2$ from $\mathcal{H}_1$. Such a reduction uses its actual adversary $\mathcal{D}$ in place of $\mathcal{S}_{\rho,\mathcal{D}}$ when calculating outputs to deliver on behalf of $\mathcal{V}$, as opposed to $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ as specified by Algorithm 3.12. Thus $\mathcal{H}_1 \approx_{\mathrm{c}} \mathcal{H}_2$.

Hybrid $\mathcal{H}_3$. This hybrid is identical to $\mathcal{H}_2$, except that $\mathcal{S}$ no longer communicates with $\mathcal{Z}$ on behalf of $\mathcal{V}$; instead $\mathcal{V}$ is instantiated as a dummy party that communicates with $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$. Upon receiving a coin request for $w \in \mathbb{W}$ from $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$, $\mathcal{S}$ forwards this request to $\mathcal{S}_{\rho,\mathcal{D}}$, and then forwards v, x (in an `accept` message), and y as well, instead of computing them itself.

Notice that since $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ computes v from (u, w) and y from (v, x) in the same way that $\mathcal{S}$ did in $\mathcal{H}_2$, the distribution of outputs from $\mathcal{V}$ to $\mathcal{Z}$ is identical between them, and the change from $\mathcal{H}_2$ to $\mathcal{H}_3$ is purely syntactical. Note also that the behavior of $\mathcal{S}$ in $\mathcal{H}_3$ is identical to that of $\mathcal{S}_{\mathsf{CompilerPV}}$. Thus

$$\begin{aligned}\mathcal{H}_2 &= \mathcal{H}_3 \\ &= \left\{ \mathrm{IDEAL}_{[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,m+n/m)]\!]_{\mathsf{PV}}, \mathcal{S}_{\mathsf{CompilerPV}}^{\mathcal{S}_{\rho,\mathcal{D}},\mathcal{A}}(\rho,f,n,u,m),\mathcal{Z}}(\lambda, z) \right\}_{\substack{\lambda,n,m\in\mathbb{N}^+ : n \geq m, \\ u\in\mathbb{V}, z\in\{0,1\}^*}}\end{aligned}$$

and by transitivity, Equation 3 holds. □

Next, we explain how to achieve the required security notion of identifiable abort with public verifiability using standard techniques.

Lemma 3.15 (Folklore: NIZK + OT + BC $\implies$ $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$)**.** *The functionality $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ can be UC-realized in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}})$-hybrid model using a constant number of sequential authenticated broadcasts and no other communication, assuming the existence of a protocol that UC-realizes $\mathcal{F}_{\mathsf{OT}}$.*

Proof Sketch. According to folklore, the following construction realizes publicly verifiable constant-round secure function evaluation with identifiable abort. The actively participating parties communicate exclusively over a single authenticated broadcast channel, using instances of the OT protocol to realize private channels between each pair (see [GKM+00]). Over this channel the parties run an instance of a constant-round MPC protocol that is secure with abort against $n - 1$ semi-malicious corruptions (i.e., secure when corrupted parties follow the protocol but may choose their own randomness and may crash). Protocols satisfying this criteria include the IPS protocol [IPS08], which is defined in the OT-hybrid model, and the BMR protocol [BMR90], which additionally requires one-way functions. Realizing the ideal OT functionality yields a protocol with

no ideal oracles, which allows us to apply a protocol compiler to it. Specifically, we apply an $\mathcal{F}_{\mathsf{NIZK}}$-based formulation of the GMW [GMW87] compiler; such a variant of GMW was previously described by Asherov et al. [AJL+12] and Cohen, shelat, and Wichs [CsW19]. This transformation yields a constant-round protocol that achieves identifiable abort and public verifiability against $n-1$ malicious corruptions. If all active participants in the transformed protocol are maliciously corrupted, then any coins required can be chosen arbitrarily by the adversary, but observing verifiers on the broadcast channel can still verify that the output is in the image of the function the protocol ostensibly computed. □

Corollary 3.16. *If there exists a protocol that UC-realizes $\mathcal{F}_{\mathsf{OT}}$ and a strongly player-replaceable round-robin protocol that UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f,n,u)]\!]_{\mathsf{PV}}$ using n sequential authenticated broadcasts and no other communication, then there is a protocol in the $(\mathcal{F}_{\mathsf{NIZK}},\mathcal{F}_{\mathsf{BC}})$-hybrid model that UC-realizes $[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,m+n/m)]\!]_{\mathsf{PV}}$ and uses $O(m+n/m)$ sequential authenticated broadcasts and no other communication. Setting $m=\sqrt{n}$ yields the efficiency result promised by the title of this paper.*

Proof. Observe that $\pi_{\mathsf{Compiler}}(\rho,n,u,m)$ requires at most $m+n/m$ sequential invocations of the $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ functionality, and involves no other communication. Thus the corollary follows from Theorem 3.9 and Lemma 3.15. □

4 A Round-Robin Protocol

In this section we present a simple protocol that meets our requirements (and therefore can be used with our compiler), which is parametric over a class of update functions that is more restrictive than the compiler demands, but nevertheless broad enough to encompass several well-known sampling problems. After presenting the protocol in Section 4.1 and proving that it meets our requirements in Section 4.2, we discuss how it can be parameterized to address three different applications: sampling structured reference strings for polynomial commitments in Section 4.3, sampling structured reference strings for zk-SNARKs in Section 4.4, and constructing verifiable mixnets in Section 4.5. We begin by defining the restricted class of update functions that our protocol supports.

Definition 4.1 (Homomorphic Update Function)**.** *A deterministic polynomial-time algorithm $f:\mathbb{V}\times\mathbb{W}\to\mathbb{V}$ is a* Homomorphic Update Function *if it satisfies:*

1. *Perfect Rerandomization: for every pair of values $v_1\in\mathbb{V}$ and $w_1\in\mathbb{W}$, $\{f(f(v_1,w_1),w_2) : w_2\leftarrow\mathbb{W}\}\equiv\{f(v_1,w_3) : w_3\leftarrow\mathbb{W}\}$. If distributional equivalence is replaced by statistical or computational indistinguishability, then the property achieved is Statistical or Computational Rerandomization, respectively.*

2. *Homomorphic Rerandomization: there exists an efficient operation $\star$ over $\mathbb{W}$ such that for every $v\in\mathbb{V}$, and every pair of values $w_1,w_2\in\mathbb{W}$,*

$$f(v,w_1\star w_2)=f(f(v,w_1),w_2).$$

Furthermore, there exists an identity value $0_{\mathbb{W}} \in \mathbb{W}$ *such that* $f(v, 0_{\mathbb{W}}) = v$.

4.1 The Protocol

Our example is straightforward: each party (in sequence) calls the update function f on the previous intermediate output to generate the next intermediate output. To achieve UC-security, the protocol must be simulatable even if f is one-way. We specify that each party uses a UC-secure NIZK to prove that it evaluated f correctly; this allows the simulator to extract the randomization witness w for f even in the presence of a malicious adversary. Specifically, we define a relation for correct evaluation for any update function f:

$$\mathcal{R}_f = \{((v_1, v_2), w) \ : \ v_2 = f(v_1, w)\}$$

We also recall the standard UC NIZK functionality, originally formulated by Groth et al.

Functionality 4.2. $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}(n)$**. NIZK for Relation** $\mathcal{R}$ **[GOS12]**

This functionality interacts with n actively participating parties denoted by $\mathcal{P}_1 \ldots \mathcal{P}_n$, and with an a-priori-unspecified number of verifiers, all designated by $\mathcal{V}$, and with the ideal adversary $\mathcal{S}$. It also has black-box access to the decider for an NP-relation, $\mathcal{R}$.

Proof: On receiving $(\texttt{prove}, \mathsf{sid}, \mathsf{ssid}, x, w)$ from $\mathcal{P}_i$ for $i \in [n]$, if $\mathcal{R}(x, w) = 0$ then ignore the message and do nothing. If $\mathcal{R}(x, w) = 1$, then send $(\texttt{prove}), \mathsf{sid}, x)$ to $\mathcal{S}$, and, upon receiving $(\texttt{proof}, \mathsf{sid}, x, \pi)$ in reply, store (sid, x, π) in memory and send $(\texttt{proof}, \mathsf{sid}, \mathsf{ssid}, \pi)$ to $\mathcal{P}_i$.

Verification: On receiving $(\texttt{verify}, \mathsf{sid}, \mathsf{ssid}, x, \pi)$ from $\mathcal{V}$, check whether a record (sid, x, π) exists in memory. If it does not, then send $(\texttt{verify}), \mathsf{sid}, x, \pi)$ to $\mathcal{S}$, and, upon receiving $(\texttt{witness}, \mathsf{sid}, x, \pi, w)$ in reply, check whether $\mathcal{R}(x, w) = 1$ and store (sid, x, π) in memory if so. Regardless, if (sid, x, π) is now in memory, then send $(\texttt{accept}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{V}$; otherwise, send $(\texttt{reject}, \mathsf{sid}, \mathsf{ssid})$.

For any particular f, there may exist an efficient bespoke proof system that realizes $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$. For example, if there is a sigma protocol for $\mathcal{R}_f$, then $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ can (usually) be UC-realized by applying the Fischlin transform [Fis05] to that sigma protocol. There are also a number of generic ways to UC-realize $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ for any polynomial-time function f [SCO+01, GOS12, CSW20]. Regardless, we give our protocol description next.

Protocol 4.3. $\pi_{\mathsf{RRSample}}(f, n, u)$**. Round-robin Sampling**

This protocol is parameterized by the number of actively participating parties $n \in \mathbb{N}^+$, by a homomorphic update function $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ (as per Definition 4.1), and by a common public input $u \in \mathbb{V}$. In addition to the actively participating parties $\mathcal{P}_p$ for $p \in [n]$, the protocol involves the ideal

functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$, and it may involve one or more observing verifiers, denoted by $\mathcal{V}$.

Sampling: Let $\mathbf{v}_0 := u$. Upon receiving $(\mathtt{sample}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i$ for $i \in [n]$ repeats the following loop for $j \in [n]$:

1. If $j = i$, $\mathcal{P}_i$ samples $\mathbf{w}_j \leftarrow \mathbb{W}$, computes $\mathbf{v}_j := f(\mathbf{v}_{j-1}, \mathbf{w}_j)$ and submits $(\mathtt{prove}, \mathsf{sid}, \mathsf{GenSID}(\mathsf{sid}, j), (\mathbf{v}_{j-1}, \mathbf{v}_j), \mathbf{w}_j)$ to $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}(n)$. Upon receiving $(\mathtt{proof}, \mathsf{sid}, \mathsf{GenSID}(\mathsf{sid}, j), \boldsymbol{\pi}_j)$ in response, $\mathcal{P}_i$ broadcasts $(\mathbf{v}_j, \boldsymbol{\pi}_j)$.[a]

2. If $j \neq i$, $\mathcal{P}_i$ waits to receive $(\hat{\mathbf{v}}_j, \boldsymbol{\pi}_j)$ from $\mathcal{P}_j$, whereupon it submits $(\mathtt{verify}, \mathsf{GenSID}(\mathsf{sid}, j), (\mathbf{v}_{j-1}, \hat{\mathbf{v}}_j), \boldsymbol{\pi}_j)$ to $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$. If $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ replies with $(\mathtt{accept}, \mathsf{sid}, \mathsf{GenSID}(\mathsf{sid}, j))$, then $\mathcal{P}_i$ assigns $\mathbf{v}_j := \hat{\mathbf{v}}_j$. If $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ replies with $(\mathtt{reject}, \mathsf{sid}, \mathsf{GenSID}(\mathsf{sid}, j))$ (or if no message is received from $\mathcal{P}_j$), then $\mathcal{P}_i$ assigns $\mathbf{v}_j := \mathbf{v}_{j-1}$.

Finally, when the loop terminates, all actively participating parties output $(\mathtt{output}, \mathsf{sid}, \mathbf{v}_n)$ to the environment.[b]

Verification: If there is an observing verifier $\mathcal{V}$, then on receiving $(\mathtt{observe}, \mathsf{sid})$ from the environment $\mathcal{Z}$, it listens on the broadcast channel and follows the instructions in Step 2 for *all* $j \in [n]$. At the end, it outputs $(\mathtt{output}, \mathsf{sid}, \mathbf{v}_n)$ to the environment.

[a] Note that when our compiler is applied to this protocol, $\mathbf{a}_j = (\mathbf{v}_j, \boldsymbol{\pi}_j)$.

[b] This implies that the "output extraction" function $\mathbf{g}_{n+1}$ described in Remark 3.3 simply returns $\mathbf{v}_n$, given the protocol transcript.

4.2 Proof of Security

We include the (straightforward) proof of security for the above protocol for completeness. After the security proof, we give a corollary concerning the application of our compiler under various generic realizations of $\mathcal{F}_{\mathsf{NIZK}}$, and then we prove a theorem stating that for the class of functions covered by Definition 4.1, the compiled protocol realizes the *original* functionality.

Theorem 4.4. *Let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be a homomorphic update function per Definition 4.1. For any $n \in \mathbb{N}^+$ and $u \in \mathbb{V}$, it holds that $\pi_{\mathsf{RRSample}}(f, n, u)$ UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)]\!]_{\mathsf{PV}}$ in the presence of a malicious adversary corrupting any number of actively participating parties in the $(\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}, \mathcal{F}_{\mathsf{BC}})$-hybrid model.*

Proof. We begin by specifying a simulator $\mathcal{S}_{\mathsf{RRSample}}$, after which we argue that

$$\begin{aligned} &\forall \mathcal{A}\ \forall \mathcal{Z}, \\ &\quad \left\{ \mathrm{REAL}_{\pi_{\mathsf{RRSample}}(f,n,u), \mathcal{A}, \mathcal{Z}}(\lambda, z) \right\}_{\lambda, n \in \mathbb{N}^+, u \in \mathbb{V}, z \in \{0,1\}^*} \\ &\equiv \left\{ \mathrm{IDEAL}_{[\![\mathcal{F}_{\mathsf{PreTrans}}(f,n,u)]\!]_{\mathsf{PV}}, \mathcal{S}_{\mathsf{RRSample}}^{\mathcal{A}}(f,n,u), \mathcal{Z}}(\lambda, z) \right\}_{\lambda, n \in \mathbb{N}^+, u \in \mathbb{V}, z \in \{0,1\}^*} \end{aligned} \tag{4}$$

Simulator 4.5. $\mathcal{S}^{\mathcal{A}}_{\mathsf{RRSample}}(f, n, u)$**. Against Any Static Corruption**

This simulator is parameterized by the number of actively participating parties $n \in \mathbb{N}^+$, by a homomorphic update function $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ (as per Definition 4.1), and by the common public input $u \in \mathbb{V}$. This simulator has black-box access to the adversary $\mathcal{A}$ who may statically corrupt any number of active participants.

Init: On initial activation for the session ID sid, $\mathcal{S}_{\mathsf{RRSample}}$ begins emulating in its head an instance of the real-world experiment for $\pi_{\mathsf{RRSample}}(f, n, u)$ for the adversary $\mathcal{A}$ (to which $\mathcal{S}_{\mathsf{RRSample}}$ has black-box access). Let this emulated experiment be referred to as SUBEXPT, and let the values of $\mathbf{v}_*$, $\hat{\mathbf{v}}_*$, $\mathbf{w}_*$, and $\boldsymbol{\pi}_*$ henceforth be defined relative to their values in this sub-experiment. Furthermore, $\mathcal{S}_{\mathsf{RRSample}}$ plays the role of the ideal oracle $\mathcal{F}^{\mathcal{R}_f}_{\mathsf{NIZK}}$ in SUBEXPT, which it does by following the code of $\mathcal{F}_{\mathsf{NIZK}}$ unless otherwise specified. $\mathcal{S}_{\mathsf{RRSample}}$ forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$ in SUBEXPT, and vice versa, and when $\mathcal{A}$ announces that it wishes to corrupt a set of parties indexed by $\mathbf{P}^* \subseteq [n]$, $\mathcal{S}_{\mathsf{RRSample}}$ corrupts the corresponding parties in its own experiment. Upon learning $\mathbf{P}^*$, if $\mathbf{P}^* \neq [n]$, then $\mathcal{S}_{\mathsf{RRSample}}$ arbitrarily chooses a single honest party index $h \in [n] \setminus \mathbf{P}^*$.

Sampling: For each round $j \in [n]$ of the protocol being run in SUBEXPT, $\mathcal{S}_{\mathsf{RRSample}}$ takes one of the following strategies, as appropriate:

1. If $\mathbf{P}^* \neq [n] \wedge j = h$, then $\mathcal{S}_{\mathsf{RRSample}}$ sends $(\mathtt{sample}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_h$, and receives $(\mathtt{unbiased}, \mathsf{sid}, \hat{\mathbf{v}}_h)$ in response. Next, on behalf of $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{S}_{\mathsf{RRSample}}$ sends $(\mathtt{prove}, \mathsf{sid}, \mathsf{GenSID}(\mathsf{sid}, h), (\mathbf{v}_{h-1}, \hat{\mathbf{v}}_h))$ to $\mathcal{A}$ in SUBEXPT and receives $(\mathtt{proof}, \mathsf{sid}, \mathsf{GenSID}(\mathsf{sid}, h), \boldsymbol{\pi}_h)$ in response. It finally broadcasts $(\hat{\mathbf{v}}_h, \boldsymbol{\pi}_h)$ to the corrupt parties in SUBEXPT on behalf of $\mathcal{P}_h$. Thereafter, if any corrupt party sends $(\mathtt{verify}, \mathsf{sid}, *, (\mathbf{v}_{h-1}, \hat{\mathbf{v}}_h), \boldsymbol{\pi}_h)$ to $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{S}_{\mathsf{RRSample}}$ replies on behalf of $\mathcal{F}_{\mathsf{NIZK}}$ with an $\mathtt{accept}$ message. If $h = n$, then $\mathcal{S}_{\mathsf{RRSample}}$ sends $(\mathtt{proceed}, \mathsf{sid}, 0_{\mathbb{W}})$ to $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$, where $0_{\mathbb{W}}$ is the identity element for $\mathbb{W}$.

2. If $\mathbf{P}^* \neq [n] \wedge j \neq h$, then $\mathcal{S}_{\mathsf{RRSample}}$ uses the code of $\mathcal{P}_j$ as specified in π_{RRSample} to compute the next message for SUBEXPT, and broadcasts this message to the corrupt parties in SUBEXPT on behalf of $\mathcal{P}_j$.

3. If $j \in \mathbf{P}^*$ and $j < n$, then $\mathcal{S}_{\mathsf{RRSample}}$ does nothing other than record the values $(\hat{\mathbf{v}}_j, \boldsymbol{\pi}_j)$ transmitted and the value $\mathbf{w}_j$ submitted to $\mathcal{F}_{\mathsf{NIZK}}$ in SUBEXPT.

4. If $j \in \mathbf{P}^*$ and $j = n$, then $\mathcal{S}_{\mathsf{RRSample}}$ iterates over $\mathbf{v}_*$, $\boldsymbol{\pi}_j$, and $\mathbf{w}_j$ and checks each corresponding set of values for consistency (as specified by the protocol). Let $\mathbf{w}'_*$ be a copy of either $\mathbf{w}_{[h+1,n]}$ (if $\mathbf{P}^* \neq [n]$) or $\mathbf{w}_{[n]}$ (if $\mathbf{P}^* = [n]$) from which any malformed rows have been removed. Let

$n' = |\mathbf{w}'|$ and $\hat{\mathbf{w}}_1 := \mathbf{w}'_1$. $\mathcal{S}_{\mathsf{RRSample}}$ computes $\hat{\mathbf{w}}_i := \mathbf{w}'_i \star \hat{\mathbf{w}}_{i-1}$ for all $i \in [n']$, after which $\mathcal{S}_{\mathsf{RRSample}}$ sends $(\texttt{proceed}, \mathsf{sid}, \hat{\mathbf{w}}_{n'})$ to $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$.

Furthermore, if at any point $\mathcal{S}_{\mathsf{RRSample}}$ receives $(\texttt{need-coin}, \mathsf{sid}, \mathbb{W})$ from $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)]\!]_{\mathsf{PV}}$, $\mathcal{S}_{\mathsf{RRSample}}$ sends $(\texttt{coin}, \mathsf{sid}, 0_{\mathbb{W}})$ in reply, where $0_{\mathbb{W}}$ is the identity element for $\mathbb{W}$.

We divide our argument into two cases: first, we address the case that the adversary corrupts no more than $n-1$ active parties. Observe that in both the real and ideal worlds, excluding the simulator's specially chosen party $\mathcal{P}_h$, all honest parties (either real or emulated) generate their messages via their protocol code; thus the distribution of these messages must be identical between the two worlds. Now consider the distribution of the message transmitted by $\mathcal{P}_h$. In the ideal world, it is calculated by the functionality as the output of f on a uniform value and common input u, whereas in the real world it is calculated by $\mathcal{P}_h$ as the output of f on a uniform value and most-recent intermediate value $\mathbf{v}_{h-1}$. By the fact that $\mathbf{v}_{h-1}$ is proven to be the result of sequential semi-malicious applications of f to u and the fact that f is rerandomizing (per Definition 4.1), the distribution of $\mathcal{P}_h$'s message in the ideal world must be indistinguishable from its message in the real world (computationally, statistically, or perfectly, depending on the flavor of the rerandomization property). We argue by inspection that the behavior of $\mathcal{F}_{\mathsf{NIZK}}$ with respect to this message is also identical. Finally consider the distribution of honest party outputs to the environment. In the real world, these are computed as the result of multiple honest and/or semi-malicious applications of f to $\mathbf{v}_h$. In the ideal world, $\mathcal{F}_{\mathsf{NIZK}}$ is used to extract the randomness associated with the semi-malicious applications, the randomness of the honest applications is generated locally by the simulator, and then the applications are collapsed using the homomorphic property of f into a single application which is evaluated by $\mathcal{F}_{\mathsf{PreTrans}}$. Thus by the homomorphic rerandomization property of f (Definition 4.1), the distribution of honest party outputs is identical in the real and ideal worlds, and it follows that the two worlds are perfectly indistinguishable if no more than $n-1$ parties are corrupt.

Second, we address the case that the adversary corrupts all n active parties, but there is an honest observing verifier. In the real world, the verifier's output is calculated as the result of a chain of semi-malicious applications of f to u. In the ideal world, $\mathcal{F}_{\mathsf{NIZK}}$ extracts the randomness associated with these applications, and as before, they are collapsed via the homomorphic property of f in order to be evaluated by $\mathcal{F}_{\mathsf{PreTrans}}$. The distributions of the output of the verifier are identical in the real and ideal worlds. It follows from this and our first case that Equation 4 holds. □

Corollary 4.6. *Let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be a homomorphic update function per Definition 4.1. For any $u \in \mathbb{V}$ and $m, n \in \mathbb{N}^+$ such that $m \leq n$, there exists a protocol in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model that UC-realizes $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, m + n/m)]\!]_{\mathsf{PV}}$ and that requires $O(m + n/m)$ sequential broadcasts and no other communication, under any of the conditions enumerated in Remark 4.7.*

Remark 4.7. *Conditions under which $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ is known to be realizable for any polynomial-time f in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model:*

1. *enhanced trapdoor permutations exist.*
2. *homomorphic trapdoor functions exist and the decisional linear assumption holds in a bilinear group.*
3. *the LWE assumption holds.*
4. *both the LPN and DDH assumptions hold.*

Proof. Applying the UC theorem to π_{RRSample} to realize $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ via the protocols of either De Santis et al. [SCO+01, CLOS02] (under the first condition), Groth et al. [GOS12] (under the second condition), or Canetti et al. [CSW20] (under the third or fourth) yields a round-robin protocol with no ideal oracle invocations in the next-message function of the active parties (though an invocation of $\mathcal{F}_{\mathsf{CRS}}$ is required ahead of time). Observe that the resulting protocol is also strongly player-replaceable, since the only inputs provided by any party random values. Since the next-message function can be represented as a circuit, our compiler can be applied to it, and so the efficiency claim follows from Corollary 3.16. □

Finally, we show that if f satisfies Definition 4.1, then the rejection-sampling capabilities the adversary has in the compiled protocol do not confer any additional power.

Theorem 4.8. *Let $f : \mathbb{V} \times \mathbb{W} \to \mathbb{V}$ be a homomorphic update function per Definition 4.1. For any value of $r \in \mathbb{N}$, the ideal-world protocol involving $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)]\!]^{\mathsf{PV}}$ perfectly UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f, n, u)]\!]^{\mathsf{PV}}$ in the presence of a malicious adversary corrupting any number of active participants.*

Proof Sketch. Rather than fully specifying a simulator, we briefly describe one and argue for the security of the ideal-world experiment from the real-world one. Note that in this proof we show that one ideal-world protocol UC-emulates another. Consequently, the simulator must internally emulate an instance of the ideal protocol for $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, n, u, r)]\!]^{\mathsf{PV}}$ in which it plays the role of the functionality (and *only* the functionality) to the ideal-world adversary for $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]^{\mathsf{PV}}$. It forwards any `sample` or `coin` messages it receives to $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ in its own experiment, and also forwards any `need-coin` messages from $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$ to the adversary in the emulated experiment on behalf of $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]^{\mathsf{PV}}$. On receiving the message $(\texttt{unbiased}, \mathsf{sid}, v)$ from $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]^{\mathsf{PV}}$, the simulator samples a value $\mathbf{w}_1 \leftarrow \mathbb{W}$ and uses it to calculate $\mathbf{v}_1 := f(v, \mathbf{w}_1)$, which is a rerandomization of v. The simulator then sends $(\texttt{candidate}, \mathsf{sid}, 1, \mathbf{v}_1)$ to the adversary in the emulated experiment. If $\mathbf{v}_1$ is rejected, then the simulator rerandomizes v again by sampling $\mathbf{w}_2 \leftarrow \mathbb{W}$ and calculating $\mathbf{v}_2 := f(v, \mathbf{w}_2)$, and then sends $\mathbf{v}_2$ as the next candidate, and so on, until the adversary in the emulated experiment accepts a candidate. Let $\mathbf{v}_k$ be the accepted candidate and $\mathbf{w}_k$ be the associated randomizing value in $\mathbb{W}$. When the adversary in the emulated experiment accepts this candidate, it sends a bias x to the simulator (in its role

as $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$), the simulator uses the homomorphic property of f to calculate a single bias $x' := \mathbf{w}_k \star x$ that combines the coins used to rerandomize the accepted candidate with the adversary's bias, and delivers this combined bias x' to $[\![\mathcal{F}_{\mathsf{PreTrans}}]\!]_{\mathsf{PV}}$. Thus in the ideal-world experiment involving $[\![\mathcal{F}_{\mathsf{PreTrans}}(f,n,u)]\!]_{\mathsf{PV}}$ and this simulator (given black-box access to an adversary), the distribution of candidates presented to the adversary is $\{f(f(u,w),\mathbf{w}_i) : w,\mathbf{w}_i \leftarrow \mathbb{W}\}$ and the output is calculated as $y = f(f(u,w),\mathbf{w}_k \star x)$ given the adversary's choice of k. On the other hand, in the ideal-world experiment involving $[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,r)]\!]_{\mathsf{PV}}$ and the same adversary the distribution of candidates presented to the adversary is $\{f(u,\mathbf{w}_i) : \mathbf{w}_i \leftarrow \mathbb{W}\}$ and the output is calculated as $y = f(f(u,\mathbf{w}_k),x)$ given the adversary's choice of k. By the perfect-rerandomization property of f, these two candidate distributions are identical, and by the homomorphic-rerandomization property of f, the output distributions are as well. Thus $[\![\mathcal{F}_{\mathsf{PostTrans}}(f,n,u,r)]\!]_{\mathsf{PV}}$ UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(f,n,u)]\!]_{\mathsf{PV}}$, regardless of the value of r. □

4.3 Application: Powers of Tau and Polynomial Commitments

In this section we specialize π_{RRSample} to the case of sampling the powers of tau, which was previously introduced in Section 1.1. Specifically, we define an update function for the powers of tau in any prime-order group $\mathbb{G}$ with maximum degree $d \in \mathbb{N}^+$ as follows:

$$\mathbb{V} = \mathbb{G}^d \qquad \mathbb{W} = \mathbb{Z}_{|\mathbb{G}|}$$
$$f : \mathbb{V} \times \mathbb{W} \to \mathbb{V} = \mathsf{PowTau}_{\mathbb{G},d}(\mathbf{V},\tau) \mapsto \left\{\tau^i \cdot \mathbf{V}_i\right\}_{i\in[d]}$$

It is easy to see that if G is a generator of $\mathbb{G}$, then $\mathsf{PowTau}_{\mathbb{G},d}(\{G\}_{i\in[d]},\tau)$ computes the powers of τ in $\mathbb{G}$ up to degree d. Proving that this function satisfies Definition 4.1 will allow us to apply our results from Section 4.2.

Lemma 4.9. *For any prime-order group $\mathbb{G}$ and any $d \in \mathbb{N}^+$, $\mathsf{PowTau}_{\mathbb{G},d}$ is a homomorphic update function with perfect rerandomization, per Definition 4.1.*

Proof. It can be verified by inspection that the homomorphic-rerandomization property of $\mathsf{PowTau}_{\mathbb{G},d}$ holds if the operator $\star$ is taken to be multiplication modulo the group order. That is, if $q = |\mathbb{G}|$, then for any $\alpha,\beta \in \mathbb{Z}_q$ and any $\mathbf{V} \in \{\mathbb{G}\}_{i\in[d]}$, we have $\mathsf{PowTau}_{\mathbb{G},d}(\mathsf{PowTau}_{\mathbb{G},d}(\mathbf{V},\alpha),\beta) = \mathsf{PowTau}_{\mathbb{G},d}(\mathbf{V},\alpha\cdot\beta \bmod q)$. If we combine this fact with the fact that $\{\mathsf{PowTau}_{\mathbb{G},d}(\mathbf{V},\tau) : \tau \leftarrow \mathbb{Z}_q\}$ is uniformly distributed over the image of $\mathsf{PowTau}_{\mathbb{G},d}(\mathbf{V},\cdot)$, then perfect rerandomization follows as well. □

As we have previously discussed, the powers of tau are useful primarily as a structured reference string for other protocols. In light of this fact, it does not make sense to construct a sampling protocol that itself requires a structured reference string. This prevents us from realizing $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_{\mathsf{PowTau}_{\mathbb{G},d}}}(n)$ via the constructions of Groth et al. [GOS12], or Canetti et al. [CSW20]. Fortunately, the NIZK

construction of De Santis et al. [SCO+01] requires only a *uniform* common random string. Thus we achieve our main theoretical result with respect to the powers of tau:

Corollary 4.10. *For any prime-order group $\mathbb{G}$ and any $d \in \mathbb{N}^+$, $n \in \mathbb{N}^+$, $m \in [n]$, and $\mathbf{V} \in \mathbb{G}^d$, there exists a protocol in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model (with a uniform CRS distribution) that UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(\mathsf{PowTau}_{\mathbb{G},d}, n, \mathbf{V})]\!]_{\mathsf{PV}}$ and that requires $O(m + n/m)$ sequential broadcasts and no other communication, under the assumption that enhanced trapdoor permutations exist.*

Proof. By conjunction of Lemma 4.9 and Theorems 4.6 and 4.8 under the restriction that the CRS distribution be uniform. □

The above corollary shows that if we set $m := \sqrt{n}$, then we can sample well-formed powers-of-tau structured reference strings with guaranteed output delivery against $n - 1$ malicious corruptions in $O(\sqrt{n})$ broadcast rounds. However, most schemes that use structured reference strings with this or similar structures assume that the strings have been sampled (in a trusted way) with *uniform* trapdoors. Our protocol does *not* achieve this, and indeed cannot without violating Cleve's lower bound [Cle86]. Instead, our protocol allows the adversary to introduce some bias. In order to use a reference string sampled by our protocol in any particular context, it must be proven (in a context-specific way) that the bias does not give the adversary any advantage.

Although previous work has proven that the bias in the reference string induced by protocols for distributed sampling can be tolerated by zk-SNARKs [BGG18, KMSV21], such proofs have thus far been monolithic and specific to the particular combination of a zk-SNARK and a sampling scheme that they address. Moreover, because zk-SNARKs are proven secure in powerful idealized models, prior distributed sampling protocols were analyzed in those models as well. Unlike zk-SNARKs, which require knowledge assumptions, the security of the Kate et al. [KZG10] polynomial-commitment scheme can be reduced to a concrete falsifiable assumption. This presents a clean, standalone context in which to examine the impact of adversarial bias in the trapdoor of a powers-of-tau reference string. We do not recall the details of the polynomial-commitment construction,[7] but note that its security follows from the d-Strong Diffie-Hellman (or d-SDH) Assumption [KZG10, Theorem 1]. We show that replacing an ideal bias-free powers-of-tau reference string with a reference string that is adversarially biased as permitted by our functionality $\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{PowTau}_{\mathbb{G},d}, n, \{G\}_{i\in[d]}, r)$ yields *no advantage* in breaking the d-SDH assumption, regardless of the value of r, so long as no more than $n - 1$ parties are corrupt. We begin by recalling the d-SDH assumption:

[7] Kate et al. actually present two related schemes. The first uses the powers of tau, exactly as we have presented it, and the second requires the powers, plus the powers again with a secret multiplicative offset (or, alternatively, relative to a second group generator). It is easy to modify our construction to satisfy the second scheme, and so for clarity we focus on the first, simpler one.

Definition 4.11 (*d*-**Strong Diffie-Hellman Assumption** [BB04])**.** *Let the security parameter λ determine a group $\mathbb{G}$ of prime order q that is generated by G. For every PPT adversary $\mathcal{A}$,*

$$\Pr\left[(c, G/(\tau+c)) = \mathcal{A}\left(\left\{\tau^i \cdot G\right\}_{i\in[d]}\right) : \tau \leftarrow \mathbb{Z}_q\right] \in \mathsf{negl}(\lambda)$$

We wish to formulate a variant of the above assumption that permits the same bias as $\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{PowTau}_{\mathbb{G},d}, n, \{G\}_{i\in[d]}, r)$. In order to do this, we define a sampling algorithm that uses the code of the functionality. We then give a formal definition of the biased assumption, which we refer to as the (n, r)-Biased d-Strong Diffie-Hellman (or (n, r, d)-SDH) assumption.

Algorithm 4.12. $\mathsf{AdvSample}^{\mathcal{Z}}_{\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{PowTau}_{\mathbb{G},d}, n, \{G\}_{i\in[d]}, r)}(1^\lambda)$

Let $\mathcal{Z}$ be a PPT adversarial algorithm that is compatible with the environment's interface to an ideal-world UC experiment involving $\mathcal{F}_{\mathsf{PostTrans}}$ and the dummy adversary $\mathcal{D}$. Let $\mathcal{Z}$ be guaranteed to corrupt no more than $n-1$ parties, and let it output some state s on termination.

1. Using the code of $\mathcal{F}_{\mathsf{PostTrans}}$, begin emulating an instance of the ideal-world experiment for $\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{PowTau}_{\mathbb{G},d}, n, \{G\}_{i\in[d]}, r)$, with $\mathcal{Z}$ as the environment. Let $\mathcal{P}_h$ be the honest party guaranteed in this experiment by the constraints on $\mathcal{Z}$.
2. In the emulated experiment, on receiving $(\mathtt{sample}, \mathsf{sid})$ from $\mathcal{Z}$ on behalf of $\mathcal{P}_h$, forward this message to $\mathcal{F}_{\mathsf{PostTrans}}$ on behalf of $\mathcal{P}_h$ as a dummy party would, and then wait to receive $(\mathtt{output}, \mathsf{sid}, z = \left\{\tau \cdot G, \tau^2 \cdot G, \ldots, \tau^d \cdot G\right\})$ for some $\tau \in \mathbb{Z}_q$ from $\mathcal{F}_{\mathsf{PostTrans}}$ in reply.
3. Extract τ from the internal state of $\mathcal{F}_{\mathsf{PostTrans}}$, and wait for $\mathcal{Z}$ to terminate with output s.
4. Output (s, τ)

Definition 4.13 ((n, r)-**Biased** d-**Strong Diffie-Hellman Assumption**)**.** *Let the security parameter λ determine a group $\mathbb{G}$ of prime order q that is generated by G. For every pair of PPT adversaries $(\mathcal{Z}, \mathcal{A})$,*

$$\Pr\left[\begin{array}{l}\mathcal{A}\left(s, \left\{\tau^i \cdot G\right\}_{i\in[d]}\right) = (c, G/(\tau+c)) \ : \\ \qquad (s,\tau) \leftarrow \mathsf{AdvSample}^{\mathcal{Z}}_{\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{PowTau}_{\mathbb{G},d}, n, \{G\}_{i\in[d]}, r)}(1^\lambda)\end{array}\right] \in \mathsf{negl}(\lambda)$$

Note that per Canetti [Can01], the dummy adversary $\mathcal{D}$ can be used to emulate any other adversary. Thus if one were to use an n-party instance of $\mathcal{F}_{\mathsf{PostTrans}}$ to generate the structured reference string for a protocol that uses the polynomial commitments of Kate et al. [KZG10], the hardness assumption that would underlie the security of the resulting scheme is (n, r, d)-SDH. We show that for all parameters n, r, the (n, r, d)-SDH assumption is *exactly* as hard as d-SDH.

Theorem 4.14. *For every $n, r, d \in \mathbb{N}^+$ and t-time adversary $(\mathcal{Z}, \mathcal{A})$ that succeeds with probability ε in the (n, r, d)-SDH experiment, there exists a t'-time adversary $\mathcal{B}$ for the d-SDH experiment that succeeds with probability ε, where $t' \approx t$.*

Proof. We begin by constructing $\mathcal{B}$:

Algorithm 4.15. $\mathcal{B}^{(\mathcal{Z},\mathcal{A})}$

1. Receive challenge $\mathbf{A}_* = \{\tau^i \cdot G\}_{i\in[d]}$ from the d-SDH challenger.
2. Emulate an instance of the (n, r, d)-SDH experiment to $(\mathcal{Z}, \mathcal{A})$, but in the code of $\mathcal{F}_{\mathsf{PostTrans}}$, act as though $u = \mathbf{A}_*$ instead of $u = \{G\}_{i\in[d]}$.
3. Let $\mathbf{w}_k \in \mathbb{Z}_q$ be the rejection-sampled randomization and $x \in \mathbb{Z}_q$ the bias selected by $\mathcal{Z}$ in its interaction with $\mathcal{F}_{\mathsf{PostTrans}}$. Let $y = \{(\mathbf{w}_k \cdot x)^i \cdot \mathbf{A}_i\}_{i\in[d]}$ be the final sampled output.
4. Once $\mathcal{Z}$ outputs its state s, compute $(c, C) \leftarrow \mathcal{A}(s, y)$
5. Output $(c/(\mathbf{w}_k \cdot x), \mathbf{w}_k \cdot x \cdot C)$

Observe that the output of $\mathcal{F}_{\mathsf{PostTrans}}$ is of the form

$$y = \left\{(\mathbf{w}_k \cdot x)^i \cdot \mathbf{A}_i\right\}_{i\in[d]} = \left\{(\mathbf{w}_k \cdot x \cdot \tau)^i \cdot G\right\}_{i\in[d]}$$

and that the views of $(\mathcal{Z}, \mathcal{A})$ induced by $\mathcal{B}$ is *identical* to their views in the (n, r, d)-SDH experiment, due to the perfect-rerandomization property of PowTau. It follows that the probability that $\mathcal{A}(s, y)$ outputs a valid solution (c, C) to the (n, r, d)-SDH problem is exactly ε. That is, with probability ε, $C = G/(\mathbf{w}_k \cdot x \cdot \tau + c)$. Thus it holds that $\mathbf{w}_k \cdot x \cdot C = G/(\tau + c/(\mathbf{w}_k \cdot x))$ and $(c/(\mathbf{w}_k \cdot x), \mathbf{w}_k \cdot x \cdot C)$ constitutes a valid solution to the d-SDH challenger with probability ε.

Note that the running time t' of $\mathcal{B}$ is only marginally more than the combined running time t of $(\mathcal{Z}, \mathcal{A})$, the overhead being due to the emulation of $\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{PowTau}_{\mathbb{G},d}, n, \{G\}_{i\in[d]}, r)$ and the adjustment in Step 5 of $\mathcal{B}$. □

4.4 Application: Sampling Updateable SRSes

In this section we discuss the specialization of our protocol to the application of sampling updateable structured reference strings for zk-SNARKs. The game-based notion of updateable security with respect to structured reference strings was defined recently by Groth et al. [GKM+18]. Informally, if a zk-SNARK has an updateable SRS, then any party can publish and update to the SRS at any time, along with a proof of well-formedness, and the security properties of the zk-SNARK hold so long as at least one honest party has contributed at some point. We direct the reader to Groth et al. for a full formal definition. Because the update operation is defined to be a local algorithm

producing a new SRS and a proof of well-formedness, which takes as input only a random tape and the previous SRS state, it is tempting to consider the protocol comprising sequentially broadcasted SRS updates by every party as a pre-existing specialization of π_{RRSample}. However, we require that the proof of well-formedness be a realization of $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ for whatever f maps the previous SRS to the next one, and the update algorithm of Groth et al. (also used by later works [MBKM19, GWC19, CHM+20]) does not have straight-line extraction. Modifying any updateable zk-SNARK to fit into our model is beyond the scope of this work. Nevertheless, we discuss two alternatives that do not involve modifying the zk-SNARK.

First, we observe that if the proofs of well-formedness of the Groth et al. update procedure [GKM+18] are taken to be part of the SRS itself, then the entire update function (let it be called $\mathsf{GrothUpdate}$) is in fact a homomorphic update procedure per Definition 4.1, by an argument similar to our proof of Lemma 4.9. This implies a result similar to Corollary 4.10: for any $n, m \in \mathbb{N}^+$ such that $m \leq n$, there exists a protocol in the uniformly distributed CRS model that UC-realizes $\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{GrothUpdate}, n, 1_{\mathsf{SRS}}, m + n/m)$ while using only $O(m + n/m)$ broadcasts under the assumption that enhanced trapdoor permutations exist, where 1_{SRS} is the "default" SRS. Furthermore, the well-formedness of SRSes generated via this protocol can be verified without checking the entire protocol transcript.

Second, we can define the functions f mapping the previous SRS to the next one (without the proofs), specialize our protocol π_{RRSample} for that function (realizing $\mathcal{F}_{\mathsf{NIZK}}^{f}$ generically), and rely on the public verifiability of $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ to ensure that the resulting SRS has the well-formedness property required. In service of this approach, we present the update functions for three recent zk-SNARKs. The update function $\mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}$ is a simple modification of $\mathsf{PowTau}_{\mathbb{G},d}$ that is compatible with both Marlin [CHM+20] and Plonk [GWC19]:

$$\mathbb{V} = \mathbb{G}_1^d \times \mathbb{G}_2 \qquad \mathbb{W} = \mathbb{Z}_q$$
$$f : \mathbb{V} \times \mathbb{W} \to \mathbb{V} = \mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}((\mathbf{X}, Y), \tau) \mapsto \left(\left\{\tau^i \cdot \mathbf{X}_i\right\}_{i\in[d]}, \tau \cdot Y\right)$$

whereas Sonic [MBKM19] has a more complex SRS with a more complex update function

$$\mathbb{V} = \mathbb{G}_1^{4d} \times \mathbb{G}_2^{4d+1} \times \mathbb{G}_{\mathsf{T}} \qquad \mathbb{W} = \mathbb{Z}_q^2 \qquad f : \mathbb{V} \times \mathbb{W} \to \mathbb{V} = \mathsf{SonicSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}$$

$$\mathsf{SonicSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}((\mathbf{X}, \mathbf{Y}, Z), (\tau, \beta))$$
$$\mapsto \begin{pmatrix} \left\{\tau^{i-d-1} \cdot \mathbf{X}_i\right\}_{i\in[d]} \| \left\{\tau^i \cdot \mathbf{X}_{i+d}\right\}_{i\in[d]} \| \left\{\beta \cdot \tau^i \cdot \mathbf{X}_{i+3d+1}\right\}_{i\in[-d,d]\setminus\{0\}}, \\ \left\{\tau^{i-d-1} \cdot \mathbf{Y}_i\right\}_{i\in[d]} \| \left\{\tau^i \cdot \mathbf{Y}_{i+d}\right\}_{i\in[d]} \| \left\{\beta \cdot \tau^i \cdot \mathbf{Y}_{i+3d+1}\right\}_{i\in[-d,d]}, \beta \cdot Z \end{pmatrix}$$

and all three have homomorphic rerandomization per Definition 4.1, by an argument similar to our proof of Lemma 4.9.

Because zk-SNARKs with updateable SRSes must tolerate adversarial updates, it seems natural to assume that they can tolerate the adversarial bias

induced by either of the above sampling methods. However, as we have mentioned, their proofs tend to be in powerful idealized models that are incompatible with UC, and so formalizing this claim is beyond the scope of this work.

4.5 Application: Verifiable Mixnets

Finally, we discuss the specialization of π_{RRSample} to the mixing procedure of verifiable mixnets. Most mixnet security definitions, whether game-based or simulation based, encompass a suite of algorithms (or interfaces, in the simulation-based case) for key generation, encryption, mixing, and decryption. We reason only about the mixing function, via an exemplar: the game-based protocol of Boyle et al. [BKRS18]. Though we do not give formal proofs, and argue that the security of the overall mixnet construction is preserved under our transformation.

Boyle et al. base their mixnet upon Bellare et al.'s [BHY09] *lossy* variant of El Gamal encryption for constant-sized message spaces. Let the message-space size be given by ϕ. Given a group $\mathbb{G}$ (chosen according to the security parameter λ) of prime order q and generated by G, it is as follows:

$$\begin{aligned}
\mathsf{KeyGen}_{\mathbb{G}}(\mathsf{sk} \in \mathbb{Z}_q) &\mapsto (\mathsf{sk}, \mathsf{pk}) : \mathsf{pk} := \mathsf{sk} \cdot G \\
\mathsf{Enc}_{\mathsf{pk}}(m \in [\phi], r \in \mathbb{Z}_q) &\mapsto (R, C) : R := r \cdot G,\ C := r \cdot \mathsf{pk} + m \cdot G \\
\mathsf{ReRand}_{\mathsf{pk}}((R, C) \in \mathbb{G}^2, r \in \mathbb{Z}_q) &\mapsto (S, D) : S := R + r \cdot G,\ D := r \cdot \mathsf{pk} + C \\
\mathsf{Dec}_{\mathsf{sk}}((R, C) \in \mathbb{G}^2) &\mapsto m \in [\phi] \ \text{ s.t. } \ m \cdot G = C + R/\mathsf{sk}
\end{aligned}$$

Note that we have given the random values (sk and r) for each function as inputs, but they must be sampled uniformly and secretly in order to prove that the above algorithms constitute an encryption scheme. Boyle et al. define the notion of a (perfectly) rerandomizable encryption scheme and assert that the above scheme satisfies it. We claim that given any $\mathsf{pk} \in \mathbb{G}$, if the homomorphic operator $\star$ is taken to be addition over $\mathbb{Z}_q$, then $\mathsf{ReRand}_{\mathsf{pk}}$ is a homomorphic update function per Definition 4.1. Given $\mathsf{ReRand}_{\mathsf{pk}}$, the ciphertext mixing function for a vector of d ciphertexts in the Boyle et al. mixnet is as follows:

$$\begin{aligned}
\mathbb{V} &= (\mathbb{G} \times \mathbb{G})^d \qquad\qquad \mathbb{W} = \Sigma_d \times \mathbb{Z}_q^d \\
f &= \mathsf{Mix}_{\mathsf{pk},d}(\mathbf{c}, (\sigma, \mathbf{r})) \mapsto \left\{\mathsf{ReRand}_{\mathsf{pk}}(\mathbf{c}_{\sigma^{-1}(i)}, \mathbf{r}_i)\right\}_{i \in [d]}
\end{aligned}$$

where Σ_d is the set of all permutations over d elements. We claim that this function is a homomorphic update function.

Lemma 4.16. *For any $\mathsf{pk} \in \mathbb{G}$ and any $d \in \mathbb{N}^+$, $\mathsf{Mix}_{\mathsf{pk},d}$ is a homomorphic update function with perfect rerandomization, per Definition 4.1.*

Proof Sketch. Perfect rerandomization holds because all elements in the vector of ciphertexts are individually perfectly rerandomized. The homomorphic operator is defined to be

$$\star : ((\sigma_1, \mathbf{r}), (\sigma_2, \mathbf{s})) \mapsto \left(\sigma_1 \circ \sigma_2, \left\{\mathbf{s}_i + \mathbf{r}_{\sigma_2^{-1}(i)}\right\}_{i \in [d]}\right)$$

where $\circ$ is the composition operator for permutations. $\square$

In the mixnet design of Boyle et al., every mixing server runs $\mathsf{Mix}_{\mathsf{pk},d}$ in sequence and broadcasts the output along with a proof that the function was evaluated correctly. In other words, their protocol is round-robin and player replaceable. Because their proofs of correct execution achieve only witness-indistinguishability (which is sufficient for their purposes), whereas we require our proofs to UC-realize $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_{\mathsf{Mix}_{\mathsf{pk},d}}}$, their protocol is not a pre-existing specialization of π_{RRSample}. Nevertheless, we can realize $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_{\mathsf{Mix}_{\mathsf{pk},d}}}$ generically as we have in our previous applications.

Corollary 4.17. *For any prime-order group $\mathbb{G}$, any $d \in \mathbb{N}^+$, any $n, m \in \mathbb{N}^+$ such that $m \leq n$, any $\mathsf{pk} \in \mathbb{G}$, and any $\mathbf{c} \in \mathrm{image}(\mathsf{Enc}_{\mathsf{pk}})^d$, there exists a protocol in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model that UC-realizes $[\![\mathcal{F}_{\mathsf{PreTrans}}(\mathsf{Mix}_{\mathsf{pk},d}, n, \mathbf{c})]\!]_{\mathsf{PV}}$ and that requires $O(m + n/m)$ sequential broadcasts and no other communication, under any of the conditions enumerated in Remark 4.7.*

Proof. By conjunction of Lemma 4.16 and Theorems 4.6 and 4.8. $\square$

We remark that the public-verifiability aspect of the functionality ensures that the mixnet that results from integrating it into the scheme of Boyle et al. is verifiable in the sense that they require [BKRS18, Definition 7]. Furthermore, the game-based security definition of Boyle et al. [BKRS18, Definition 12] permits the adversary to induce *precisely* the same sort of bias as $[\![\mathcal{F}_{\mathsf{PreTrans}}(\mathsf{Mix}_{\mathsf{pk},d}, \cdot, \cdot)]\!]_{\mathsf{PV}}$. It follows naturally that their construction retains its security properties when mixing is done via our functionality. Setting $m := \sqrt{n}$, we have achieved a verifiable mixnet with guaranteed output delivery against $n - 1$ maliciously-corrupt mix servers in $O(\sqrt{n})$ broadcast rounds.

Finally, we note that the above transformation can be applied to other mixnets as well. Consider the UC-secure mixnet protocol of Wikström [Wik05]. It also uses a variation of El Gamal encryption, but it combines mixing and threshold decryption into a single protocol phase, which makes it incompatible with our transformation as written. If the two operations are separated, then our transformation can be applied to the mixing phase to reduce it from n rounds to $O(\sqrt{n})$, just as with Boyle et al.

5 With Concrete Efficiency

The previous sections of this paper have been concerned with optimizing round efficiency to the exclusion of all else. In practice, this may lead to protocols that are concretely round-efficient, but prohibitively expensive due to large concrete communication or computation costs. Consider, for example, the powers of tau in an elliptic curve: in practice, $d \in [2^{10}, 2^{20}]$ [BGM17]. This implies that the $\mathsf{PowTau}_{\mathbb{G},d}$ function involves thousands (possibly *many* thousands) of elliptic-curve scalar multiplications; if rendered into a Boolean circuit, the function could easily require trillions of gates. Evaluating circuits of such size is at or

beyond the edge of feasibility with current techniques even in the security-with-abort setting, and our compiler requires the circuit to be evaluated many times with identifiable abort.

We believe that this concrete inefficiency is a shortcoming of our *compiler* and not the technique that underlies it. In evidence of this, we use this section to sketch a new protocol, $\pi_{\mathsf{BilinearSRS}}$, which realizes $\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}, n, (\mathbf{X}, Y), 2\sqrt{n}-1)$ *directly*, where $(\mathbf{X}, Y)$ is any well-formed SRS. Our new protocol requires $O(\sqrt{n} \cdot \log d)$ sequential broadcast rounds and avoids the major concrete costs implied by compiling the round-robin protocol. For details of the BilinearSRS update function for bilinear groups, and its applications to zk-SNARKs, we refer the reader to Section 4.4, and we note that our construction could easily be adapted to sample with related update functions such as PowTau. We begin with a simple and intuitive description, and then give the full formal protocol in Section 5.1, a concrete cost analysis in Section 5.2, and a proof of security in Section 5.3.

$\pi_{\mathsf{BilinearSRS}}$ will leverage the fact that the well-formedness of SRSes sampled by the BilinearSRS update function can be checked using the pairing operation of the underlying bilinear group, without any additional protocol artifacts or external information. $\pi_{\mathsf{BilinearSRS}}$ is structured similarly to π_{Compiler}, with two major differences. First, when a committee's intermediate output is chosen to become definitive, it is first double-checked for well-formedness by *all* parties in the protocol (the check is performed via the pairing operation and therefore incurs only computational costs), and the entire committee is ejected for cheating if this check fails. Second, we replace instances of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ that evaluate the $\mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}$ update function as a circuit with instances of a new *SRS extension* functionality $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ that directly computes the same update function. $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ maintains most of the public-verifiability properties of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$, but unlike the latter it allows the adversary to choose the output arbitrarily if all active participants are corrupted (which necessitates the aforementioned double-check).

In order to realize $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ with reasonable concrete efficiency, each committee samples shares of a uniform secret τ and uses a generic *reactive arithmetic* MPC functionality $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ to compute secret sharings of the powers of τ. The functionality $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ is similar to $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$, except that it is *reactive* (that is, it allows the circuit to be determined dynamically after inputs are supplied), it allows the adversary to choose outputs arbitrarily if all active participants are corrupted (much like $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$), and it natively supports arithmetic computations over an arbitrary field, which implies that this computation requires only $O(d)$ gates arranged in a circuit of depth $O(\log d)$. Using these shares of the powers of τ, the committee engages in a round of distributed scalar operations to generate its intermediate SRS, which is checked for well-formedness by the members of the committee (but *not* by any passive verifiers). If any active participants are honest, and the intermediate SRS is not well-formed, then they broadcast a message indicating as much, along with information that allows passive verifiers to efficiently confirm which active participant has cheated. Known techniques [BOS16] for realizing $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$

require a round count proportionate to the circuit's multiplicative depth, and so the protocol realizing $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ runs in $O(\log d)$ rounds overall.

In practice, bilinear groups are realized by certain elliptic curves, and the pairing operation has a large concrete computational cost; thus we must use it judiciously. The elliptic-curve scalar multiplication operation is also expensive, and so we wish to minimize the use of this operation as well. In our protocol, these two operations incur the vast majority of computational costs that are not due to the protocol realizing $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$. In Section 5.2 we will argue that for our protocol, the latency incurred by these operations is both asymptotically and concretely less than the latency incurred by the same operations in the state of the art round-robin sampling protocol.

5.1 Protocols

Now we give the formal description of the concretely efficient protocol for SRS sampling described previously. We begin by specifying the functionality $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ that the committees use for computing SRS updates.

Functionality 5.1. $\mathcal{F}_{\mathsf{ExtSRS}}(n, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)$**. SRS Extension**

This functionality interacts with n actively participating parties denoted by $\mathcal{P}_1 \ldots \mathcal{P}_n$ and with the ideal adversary $\mathcal{S}$. It is also parameterized by a pair of prime-order groups $\mathbb{G}_1, \mathbb{G}_2$ such that $|\mathbb{G}_1| = |\mathbb{G}_2| = q$, a vector of group elements $\mathbf{X} \in \mathbb{G}_1^d$ for some $d \in \mathbb{N}^+$, and a single group element $Y \in \mathbb{G}_2$.

Extension Sampling: On receiving $(\texttt{sample}, \mathsf{sid})$ from at least one party $\mathcal{P}_i$ for $i \in [n]$,

1. If $(\texttt{sample}, \mathsf{sid})$ has been received from at least one active participant, sample $\tau \leftarrow \mathbb{Z}_q$ and then compute $\mathbf{A} := \{\tau^i \cdot \mathbf{X}_i\}_{i \in [d]}$ and $B := \tau \cdot Y$
2. Send $(\texttt{candidate-output}, \mathsf{sid}, (\mathbf{A}, B))$ to $\mathcal{S}$, and receive $(\texttt{proceed}, \mathsf{sid})$ or $(\texttt{override}, \mathsf{sid}, (\mathbf{A}', \tau'))$ or $(\texttt{abort}, \mathsf{sid})$ from $\mathcal{S}$ in response.
   (a) If `proceed` is received, then send $(\texttt{output}, \mathsf{sid}, (\mathbf{A}, B))$ to all parties.
   (b) If `override` is received and there are any honest parties, then ignore the message and wait for another response.
   (c) If `override` is received and there are no honest parties, then send $(\texttt{output}, \mathsf{sid}, (\mathbf{A}', \tau' \cdot Y))$ to all parties.
   (d) If `abort` is received, then send $(\texttt{secret}, \mathsf{sid}, \tau)$ to $\mathcal{S}$, receive $(\texttt{stooge}, \mathsf{sid}, c)$ in response, where c is the index of a corrupt party, and then send $(\texttt{abort}, \mathsf{sid}, c)$ to all parties.

Notice that $[\![\mathcal{F}_{\mathsf{ExtSRS}}(n, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$ behaves similarly to an instance of

$[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}(\gamma, n)]\!]_{\mathsf{PV}}$ where

$$\gamma\left(\boldsymbol{\omega} \in \mathbb{Z}_q^n\right) \mapsto \mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,|\mathbf{X}|}\Big((\mathbf{X}, Y), \sum_{i\in[n]} \boldsymbol{\omega}_i\Big)$$

so long as there is at least one honest party, who supplies a random value for its share in $\boldsymbol{\omega}$. Using this functionality, we specify a protocol to realize $\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}, n, (\mathbf{X}, Y), 2\sqrt{n}-1)$. As we have said, this protocol is extremely similar to the compiler, except for the use of $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ and the fact that all parties double-check the well-formedness of all intermediate values.

Protocol 5.2. $\pi_{\mathsf{BilinearSRS}}(n, m, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)$**. SRS Sampling**

This protocol is parameterized by the number of actively participating parties, $n \in \mathbb{N}^+$. It is also parameterized by a pair of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, both of the same prime order q, such that there exists a third group $\mathbb{G}_\mathsf{T}$ of order q and an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$. Let G_1 and G_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$. In addition, it is parameterized by a vector of group elements $\mathbf{X} \in \mathbb{G}_1^d$ for some $d \in \mathbb{N}^+$ and a single group element $Y \in \mathbb{G}_2$ with the constraint that there exists some $x \in \mathbb{Z}_q$ such that $\mathbf{Y} = x \cdot G_2$ and $\mathbf{X}_i = x^i \cdot G_1$ for every $i \in [d]$. In addition to the actively participating parties $\mathcal{P}_i$ for $i \in [n]$, the protocol involves the ideal functionality $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and it may involve one or more observing verifiers, denoted by $\mathcal{V}$.

Sampling: Let $\mathbf{A}_{0,*} := \mathbf{X}$ and $\mathbf{B}_0 := Y$ and let $\mathbf{C}_{1,*,*}$ be a deterministic partitioning of $[n]$ into m balanced subsets. That is, for $i \in [m]$, let $\mathbf{C}_{1,i,*}$ be a vector indexing the parties in the i^{th} committee. Upon receiving $(\mathtt{sample}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party repeats the following sequence of steps, starting with $k := 1$ and $\mathbf{j}_1 := 1$, incrementing k with each loop, and terminating the loop when $\mathbf{j}_k > m$

1. For all $i \in [m]$ (in parallel) each party $\mathcal{P}_\ell$ for $\ell \in \mathbf{C}_{k,i,*}$ sends $(\mathtt{sample}, \mathsf{GenSID}(\mathsf{sid}, k, i))$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}(|\mathbf{C}_{k,i,*}|, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$.
2. For all $i \in [m]$ (in parallel) each party $\mathcal{P}_\ell$ for $\ell \in [n] \setminus \mathbf{C}_{k,i,*}$ sends $(\mathtt{observe}, \mathsf{GenSID}(\mathsf{sid}, k, i))$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}(|\mathbf{C}_{k,i,*}|, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$ (thereby taking the role of verifier).
3. For $i \in [m]$, all parties receive either $(\mathtt{abort}, \mathsf{GenSID}(\mathsf{sid}, k, i), \mathbf{c}_{k,i})$ or $(\mathtt{output}, \mathsf{GenSID}(\mathsf{sid}, k, i), (\hat{\mathbf{A}}_{k,i,*}, \hat{\mathbf{B}}_{k,i}))$ from $[\![\mathcal{F}_{\mathsf{ExtSRS}}(|\mathbf{C}_{k,i,*}|, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$. In the latter case, let $\mathbf{c}_{k,i} := \bot$.
4. If any outputs were produced in the previous step, let ℓ be the smallest integer such that $(\mathtt{output}, \mathsf{GenSID}(\mathsf{sid}, k, \ell), (\hat{\mathbf{A}}_{k,\ell,*}, \hat{\mathbf{B}}_{k,\ell}))$ was received.

All parties individually sample local values $\mathbf{r} \leftarrow \mathbb{Z}_q^d$ and check that

$$e\Big(\sum_{j\in[d]} \mathbf{r}_j \cdot \hat{\mathbf{A}}_{k,\ell,j},\ G_2\Big) = e\Big(\mathbf{r}_1 \cdot G_1 + \sum_{j\in[2,d]} \mathbf{r}_j \cdot \hat{\mathbf{A}}_{k,\ell,j-1},\ \hat{\mathbf{B}}_{k,\ell}\Big)$$

and if this relation holds, then the parties set $\mathbf{j}_{k+1} := \mathbf{j}_k + 1$ and $(\mathbf{A}_{\mathbf{j}_k+1,*}, \mathbf{B}_{\mathbf{j}_k+1}) := (\hat{\mathbf{A}}_{k,\ell,*}, \hat{\mathbf{B}}_{k,\ell})$. On the other hand, if the check fails, then the parties set $\mathbf{j}_{k+1} := \mathbf{j}_k$. Regardless, for every $i \in [m]$ let

$$\mathbf{C}_{k+1,i,*} := \begin{cases} \mathbf{C}_{k,i,*} \setminus \{\mathbf{c}_{k,i}\} & \text{if } i \neq \ell \\ \varnothing & \text{if } i = \ell \end{cases}$$

5. If no outputs were produced in Step 3, then let $\mathbf{j}_{k+1} := \mathbf{j}_k$ and for every $i \in [m]$ let
$$\mathbf{C}_{k+1,i,*} := \mathbf{C}_{k,i,*} \setminus \{\mathbf{c}_{k,i}\}$$

Finally, each party outputs $(\mathtt{output}, \mathsf{sid}, (\mathbf{A}_{m,*}, \mathbf{B}_m))$ to the environment when the loop terminates.

Verification: If there is an observing verifier $\mathcal{V}$, then upon receiving $(\mathtt{observe}, \mathsf{sid})$ from the environment $\mathcal{Z}$, it repeats the following sequence of steps, starting with $k := 1$ and $\mathbf{j}_1 := 1$, incrementing k with each loop, and terminating the loop when $\mathbf{j}_k > m$.

6. $\mathcal{V}$ sends $(\mathtt{observe}, \mathsf{GenSID}(\mathsf{sid}, k, i))$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}(|\mathbf{C}_{k,i,*}|, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$ for all $i \in [m]$, and receives either $(\mathtt{abort}, \mathsf{GenSID}(\mathsf{sid}, k, i), \mathbf{c}_{k,i})$ or $(\mathtt{output}, \mathsf{GenSID}(\mathsf{sid}, k, i), (\hat{\mathbf{A}}_{k,i,*}, \hat{\mathbf{B}}_{k,i}))$ in response.

7. $\mathcal{V}$ determines the value of $\mathbf{j}_{k+1}$ and $\mathbf{C}_{k+1,*,*}$ per the method in Steps 4 and 5.

Finally, $\mathcal{V}$ outputs $(\mathtt{output}, \mathsf{sid}, (\mathbf{A}_{m,*}, \mathbf{B}_m))$ to the environment when the loop terminates.

Realizing $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$. Before we show how this functionality is to be realized, we will specify a few building blocks that are required. We begin by giving a functionality for reactive arithmetic MPC with Identifiable Abort. This functionality is a derivative of the one given by Baum et al. [BOS16].

Functionality 5.3. $\mathcal{F}_{\mathsf{MPC\text{-}IA}}(n, \mathbb{F})$. Reactive MPC with IA [BOS16]

This functionality interacts with n actively participating parties denoted by $\mathcal{P}_1 \ldots \mathcal{P}_n$ and with the ideal adversary $\mathcal{S}$. It is also parameterized by the description of a field $\mathbb{F}$ over which it operates. Whenever this functionality

receives a message from any party, it notifies the other parties that the message was received, including in the notification the sender's ID, the session ID, and any subsession IDs.

Input: On receiving $(\texttt{input}, i, \mathsf{sid}, \mathsf{ssid}, x)$ for $x \in \mathbb{F}$ from $\mathcal{P}_i$ and $(\texttt{input}, i, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{P}_j$ for every $j \in [n] \setminus \{i\}$, if ssid is a fresh subsession ID, then store $(\mathsf{sid}, \mathsf{ssid}, x)$ in memory.

Addition: On receiving $(\texttt{add}, \mathsf{sid}, \mathsf{ssid}_1, \mathsf{ssid}_2, \mathsf{ssid}_3)$ from $\mathcal{P}_i$ for every $i \in [n]$, if ssid_3 is a fresh subsession ID and records of the form $(\mathsf{sid}, \mathsf{ssid}_1, x)$ and $(\mathsf{sid}, \mathsf{ssid}_2, y)$ are stored in memory, then store $(\mathsf{sid}, \mathsf{ssid}_3, x + y)$ in memory.

Multiplication: On receiving $(\texttt{mult}, \mathsf{sid}, \mathsf{ssid}_1, \mathsf{ssid}_2, \mathsf{ssid}_3)$ from $\mathcal{P}_i$ for every $i \in [n]$, if ssid_3 is a fresh subsession ID and records of the form $(\mathsf{sid}, \mathsf{ssid}_1, x)$ and $(\mathsf{sid}, \mathsf{ssid}_2, y)$ are stored in memory, then store $(\mathsf{sid}, \mathsf{ssid}_3, x \cdot y)$ in memory.

Output: On receiving $(\texttt{output}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{P}_i$ for every $i \in [n]$, if at least one actively participating party is honest and a record of the form $(\mathsf{sid}, \mathsf{ssid}, x)$ exists in memory, then send $(\texttt{candidate-output}, \mathsf{sid}, \mathsf{ssid}, x)$ to $\mathcal{S}$, and receive $(\texttt{abort-stooge}, \mathsf{sid}, \mathsf{ssid}, c)$ in response. If c is the index of a corrupt party, then send $(\texttt{abort}, \mathsf{sid}, c)$ to all parties, and send $(\texttt{output}, \mathsf{sid}, \mathsf{ssid}, x)$ to all parties otherwise. If there are no honest actively participating parties, then send $(\texttt{adv-output}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$, and if $\mathcal{S}$ replies with a message of the form $(\texttt{output}, \mathsf{sid}, \mathsf{ssid}, *)$ or $(\texttt{abort}, \mathsf{sid}, *)$, then forward the reply to all parties.

In addition to the above functionality, we will use a commitment functionality $\mathcal{F}_{\mathsf{Com}}$ that is similar to the standard functionality given by Canetti and Fischlin [CF01]. However, since we only need "broadcast-style" (i.e. one-to-everyone) commitments, we will omit explicit destination parties, instead insisting that every commitment and decommitment is received by all participants (and any observing verifiers in the case of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$). We will also use a one-to-everyone version of the standard $\mathcal{F}_{\mathsf{CP}}$ zero-knowledge commit-and-prove functionality. Both of these functionalities are defined and discussed by Canetti et al. [CLOS02]. $\mathcal{F}_{\mathsf{CP}}$ is is parameterized by a relation, and the one that our protocol requires is the discrete logarithm relation in an elliptic-curve group:

$$\mathcal{R}_{\mathsf{DL}} = \{((X, B), x) \ : \ X = x \cdot B\}$$

Using these building blocks, we now specify a protocol to realize $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$.

Protocol 5.4. $\pi_{\mathsf{ExtSRS}}(n, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)$**. SRS Extension**

This protocol is parameterized by the number of actively participating parties, $n \in \mathbb{N}^+$. It is also parameterized by a pair of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, both of the same prime order q, such that there exists a third group $\mathbb{G}_{\mathsf{T}}$ of order q and an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_{\mathsf{T}}$. Let G_1 and G_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$. In addition, it is parameterized by a vector of group

elements $\mathbf{X} \in \mathbb{G}_1^d$ for some $d \in \mathbb{N}^+$ and a single group element $Y \in \mathbb{G}_2$ with the constraint that there exists some $x \in \mathbb{Z}_q$ such that $\mathbf{Y} = x \cdot G_2$ and $\mathbf{X}_i = x^i \cdot G_1$ for every $i \in [d]$. In addition to the actively participating parties $\mathcal{P}_i$ for $i \in [n]$, the protocol involves the ideal functionalities $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, and $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$, and it may involve one or more observing verifiers, denoted by $\mathcal{V}$.

Extension Sampling: On receiving $(\mathtt{sample}, \mathsf{sid})$ from the environment, each party $\mathcal{P}_i$ for $i \in [n]$ follows the steps below. If at any point in this sequence of steps $\mathcal{P}_i$ expects to receive a broadcast from some $\mathcal{P}_c$ or a notification from $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ that it was activated by $\mathcal{P}_c$ in a particular way, but no such broadcast or notification arrives, or if at any point $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ sends $(\mathtt{abort}, \mathsf{sid}, c)$ to $\mathcal{P}_i$, then $\mathcal{P}_i$ aborts by sending $(\mathtt{abort}, \mathsf{sid}, c)$ to the environment.

1. Each party $\mathcal{P}_i$ samples $\boldsymbol{\tau}_i \leftarrow \mathbb{Z}_q$ and $\boldsymbol{\mu}_{i,*} \leftarrow \mathbb{Z}_q^d$, and then the parties use $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}(n, \mathbb{Z}_q)]\!]_{\mathsf{PV}}$ to compute $\boldsymbol{\alpha} \in \mathbb{Z}_q^d$ such that

$$\boldsymbol{\alpha} = \Big\{ \sum_{i \in [n]} \boldsymbol{\mu}_{i,j} + \Big(\sum_{i \in [n]} \boldsymbol{\tau}_i \Big)^j \Big\}_{j \in [d]}$$

   and then $\boldsymbol{\alpha}$ is revealed to all parties. Using the following methodology, the parties can perform this procedure using $\log_2 d + 1$ batches of parallel calls to the `mult` interface of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, plus one batch of parallel calls to the `input` interface and one to the `output` interface. The parties send instructions to the functionality in batches as specified, and do not proceed until all the instructions in a batch are confirmed to have been received and evaluated. If some party $\mathcal{P}_c$ fails to send a specified instruction, then the other parties (and observing verifiers) will receive no notification from the functionality (as they would had $\mathcal{P}_c$ behaved honestly), and they will abort, identifying $\mathcal{P}_c$ as the cheater.

   (a) The parties use the `input` interface of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ to load all elements of $\boldsymbol{\tau}$ and $\boldsymbol{\mu}$ into the functionality (in parallel), and then via the `add` interface they instruct the functionality to compute

$$\tau := \sum_{i \in [n]} \boldsymbol{\tau}_i$$

   (b) Via the `mult` interface of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, the parties instruct the functionality to compute the parity-1 powers of τ, which is to say, τ^{2^k} for $k \in [\lfloor \log_2 d \rfloor]$. This requires $\lfloor \log_2 d \rfloor$ sequential invocations of the `mult` interface.

(c) As soon as the functionality has calculated each value τ^{2^k} for $k \in [\lfloor \log_2 d \rfloor]$ (as instructed in Step 1b), the parties use the `mult` interface of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ to instruct the functionality to compute the product of τ^j and τ^{2^k} for every $j < 2^k$, in parallel. The former set of values will have already been computed and stored by the functionality, and the result will be that the functionality computes and stores the values $\tau^{j'}$ for every $j' < 2^{k+1}$. This step requires $\lfloor \log_2 d \rfloor$ sequential batched invocations of the `mult` interface; since all but one of these invocations can be performed simultaneously with the invocations in Step 1b, it adds only one batch of invocations overall.

(d) The parties use the `add` interface to instruct $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ to compute

$$\boldsymbol{\alpha}_j := \tau^j + \sum_{i \in [n]} \boldsymbol{\mu}_{i,j}$$

for every $j \in [d]$, in parallel.

(e) Finally, the parties instruct $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ via the `output` interface to reveal $\boldsymbol{\alpha}$ to all parties and observing verifiers.

2. Upon being notified by $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ that all operations in the previous step were completed successfully, every $\mathcal{P}_i$ for $i \in [n]$ computes

$$\mathbf{M}_{i,*} := \left\{ (q - \boldsymbol{\mu}_{i,j}) \cdot \mathbf{X}_j \right\}_{j \in [d]} \quad \text{and} \quad \mathbf{N}_i := (q - \boldsymbol{\mu}_{i,1}) \cdot Y$$

after which it sends $(\texttt{commit}, \mathsf{GenSID}(\mathsf{sid}, i), \mathbf{M}_{i,*})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, sends $(\texttt{commit}, \mathsf{GenSID}(\mathsf{sid}, i), q - \boldsymbol{\mu}_{i,1})$ to $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$, and registers to be an observing verifier for every other party's matching instances of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$. Upon being notified that all other parties are committed, $\mathcal{P}_i$ sends $(\texttt{open}, \mathsf{GenSID}(\mathsf{sid}, i))$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and $(\texttt{prove}, \mathsf{GenSID}(\mathsf{sid}, i), (\mathbf{N}_i, Y))$ to $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$. If any party $\mathcal{P}_c$ fails to commit, open, or prove in this step, then $\mathcal{P}_i$ aborts by sending $(\texttt{abort}, \mathsf{sid}, c)$ to the environment.

3. On receiving $\mathbf{M}_{i',*}$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and $\mathbf{N}_{i'}$ from $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ for every $i' \in [n] \setminus \{i\}$, $\mathcal{P}_i$ computes

$$\mathbf{A} := \left\{ \boldsymbol{\alpha}_j \cdot \mathbf{X}_j + \sum_{i' \in [n]} \mathbf{M}_{i',j} \right\}_{j \in [d]} \quad \text{and} \quad B := \boldsymbol{\alpha}_1 \cdot Y + \sum_{i' \in [n]} \mathbf{N}_{i'}$$

4. $\mathcal{P}_i$ samples a local value $\mathbf{r} \leftarrow \mathbb{Z}_q^d$, and checks that

$$e\Big(\sum_{j \in [d]} \mathbf{r}_j \cdot \mathbf{A}_j,\ G_2 \Big) = e\Big(\mathbf{r}_1 \cdot G_1 + \sum_{j \in [2,d]} \mathbf{r}_j \cdot \mathbf{A}_{j-1},\ B \Big)$$

If this relation holds, then $\mathcal{P}_i$ broadcasts $(\texttt{ok}, \mathsf{sid}, (\mathbf{A}, B))$, and if it does not, then $\mathcal{P}_i$ broadcasts $(\texttt{not-ok}, \mathsf{sid})$. In the former case, we will say that $\mathcal{P}_i$'s *certified output value* is $(\mathbf{A}, B)$, and in the latter case, it is $\perp$.

5. On receiving `ok` or `not-ok` messages from the other active participants (certifying their outputs),
   (a) If all parties (including $\mathcal{P}_i$) certified identical non-$\perp$ outputs, then $\mathcal{P}_i$ outputs $(\texttt{output}, \mathsf{sid}, (\mathbf{A}, B))$ to the environment, and halts.
   (b) If some party $\mathcal{P}_c$ certified an output $(\mathbf{A}', B')$ such that $B' \neq B$, then $\mathcal{P}_i$ outputs $(\texttt{abort}, \mathsf{sid}, c')$ to the environment and halts. If multiple parties certified such outputs, $\mathcal{P}_c$ is taken to be the lowest-indexed among them.
   (c) If there were two parties who certified two distinct non-$\perp$ outputs $(\mathbf{A}', B)$ and $(\mathbf{A}'', B)$, then let c' and c'' their indices, and let them be the lowest such indices if more than one pair of parties meets this condition. $\mathcal{P}_i$ finds the minimal value of $j \in [d]$ such that $\mathbf{A}'_j \neq \mathbf{A}''_j$, and outputs $(\texttt{abort}, \mathsf{sid}, c')$ to the environment if $\mathbf{A}'_j \neq \mathbf{A}_j$, or else outputs $(\texttt{abort}, \mathsf{sid}, c'')$ if $\mathbf{A}'_j = \mathbf{A}_j$. $\mathcal{P}_i$ halts regardless.
   (d) Otherwise, it must be the case that at least one party certified $\perp$ and all other parties certified identical outputs. In this case, $\mathcal{P}_i$ initiates cheater identification by jumping to Step 6.

Cheater Identification: $\mathcal{P}_i$ arrives at this set of instructions if it detected a cheat elsewhere in the protocol, but the cheater was not identified.

6. The parties use the `output` interface of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}(n, \mathbb{Z}_q)]\!]_{\mathsf{PV}}$ to instruct the functionality to reveal all elements of $\boldsymbol{\tau}$ and $\boldsymbol{\mu}$ (in parallel) to all parties and observing verifiers.

7. Let $\mathbf{I} \subseteq [n]$ be a set (stored in ascending order) indexing the parties who broadcasted $(\texttt{not-ok}, \mathsf{sid})$ and thereby initiated cheater identification. $\mathcal{P}_{\mathbf{I}_1}$ (the lowest-indexed party in this set) finds the smallest value of $j \in [d]$ such that
$$\mathbf{A}_j \neq \Big(\sum_{i \in [n]} \boldsymbol{\tau}_i \Big)^j \cdot \mathbf{X}_j$$
and then finds the smallest value $c \in [n]$ such that $\mathbf{M}_{c,j} \neq \big(q - \boldsymbol{\mu}_{c,j}\big) \cdot \mathbf{X}_j$, then broadcasts $(\texttt{malformed-output}, \mathsf{sid}, c, j)$ to all parties. If no such values exist, then $\mathcal{P}_{\mathbf{I}_1}$ finds the smallest value $c \in [n]$ such that $\mathbf{N}_c \neq \big(q - \boldsymbol{\mu}_{c,1}\big) \cdot Y$ and broadcasts $(\texttt{malformed-output}, \mathsf{sid}, c, 0)$ to all parties.[a]

8. On receiving the message $(\texttt{malformed-output}, \mathsf{sid}, c, j)$ from $\mathcal{P}_{\mathbf{I}_1}$, $\mathcal{P}_i$ for $i \in [n] \setminus \{\mathbf{I}_1\}$ checks whether

$$\mathbf{M}_{c,j} = \left(q - \boldsymbol{\mu}_{c,j}\right) \cdot \mathbf{X}_j$$

If this relation holds, then $\mathcal{P}_i$ aborts by sending $(\texttt{abort}, \mathsf{sid}, \mathbf{I}_1)$ to the environment, and if this relation does not hold, then $\mathcal{P}_i$ aborts by sending $(\texttt{abort}, \mathsf{sid}, c)$ to the environment. $\mathcal{P}_{\mathbf{I}_1}$ aborts by sending $(\texttt{abort}, \mathsf{sid}, c)$ to the environment, regardless.

Verification: If there is an observing verifier $\mathcal{V}$, then upon receiving $(\texttt{observe}, \mathsf{sid})$ from the environment $\mathcal{Z}$, $\mathcal{V}$ sends $(\texttt{observe}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]^{\mathsf{PV}}$ and also begins observing the relevant instances of $[\![\mathcal{F}_{\mathsf{Com}}]\!]^{\mathsf{PV}}$ and $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]^{\mathsf{PV}}$. Thereafter it receives the same notifications and outputs as the active participants. In particular, it receives the output values in Steps 1e and 6 and the broadcasts in Steps 2, 4, and 7. It does *not* perform the randomized well-formedness check in Step 4: instead,

9. If some party $\mathcal{P}_c$ omitted an expected message anywhere in the protocol, then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, c)$ to the environment. If multiple parties omitted expected messages at the same time, then $\mathcal{P}_c$ is taken to be the lowest-indexed one.

10. If all active participants certified a consistent non-$\perp$ output $(\mathbf{A}, B)$, then $\mathcal{V}$ takes them at their word and outputs $(\texttt{output}, \mathsf{sid}, (\mathbf{A}, B))$ to the environment, and halts.

11. $\mathcal{V}$ computes

$$B \coloneqq \boldsymbol{\alpha}_1 \cdot Y + \sum_{i \in [n]} \mathbf{N}_i$$

and if any party $\mathcal{P}_c$ certified an output $(\mathbf{A}', B')$ such that $B' \neq B$, then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, c)$ to the environment and halts. If multiple parties certified such outputs, then $\mathcal{P}_c$ is taken to be the lowest-indexed one.

12. If there were two parties who certified two distinct non-$\perp$ outputs $(\mathbf{A}', B)$ and $(\mathbf{A}'', B)$, then let c' and c'' their indices, and let them be the lowest such indices if more than one pair of parties meets this condition. $\mathcal{V}$ finds the minimal value of $j \in [d]$ such that $\mathbf{A}'_j \neq \mathbf{A}''_j$, computes

$$\mathbf{A}_j \coloneqq \boldsymbol{\alpha}_j \cdot \mathbf{X}_j + \sum_{i \in [n]} \mathbf{M}_{i,j}$$

and outputs $(\texttt{abort}, \mathsf{sid}, c')$ to the environment if $\mathbf{A}'_j \neq \mathbf{A}_j$, or $(\texttt{abort}, \mathsf{sid}, c'')$ if $\mathbf{A}'_j = \mathbf{A}_j$. $\mathcal{V}$ halts regardless.

13. Otherwise, it must be the case that at least one party certified $\perp$ and all other parties certified identical outputs. Let $\mathcal{P}_i$ be the lowest-indexed party who certified $\perp$, and let $(\texttt{malformed-output}, \mathsf{sid}, c, j)$ be subsequent message sent by $\mathcal{P}_i$ in Step 7. $\mathcal{V}$ uses the relations in Step 8 to verify that $\mathcal{P}_c$ has indeed cheated. If $\mathcal{P}_c$ has cheated, then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, c)$ to the environment; otherwise it outputs $(\texttt{abort}, \mathsf{sid}, i)$. It halts regardless.

[a] All parties could perform this step, but it would improve neither asymptotic nor worst-case concrete performance.

Finally, we note that when there is only *one* active participant, the above protocol can be dramatically simplified: the single active party need not invoke $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ or perform any distributed SRS assembly or cheater identification: it simply samples τ, computes $(\mathbf{A}, B)$ locally, broadcasts its output, and uses $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ to prove knowledge of the discrete logarithm of B with respect to Y. This protocol is very similar to $\pi_{\mathsf{RRSample}}(\mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}, 1, \cdot)$, apart from the fact that the invocation of $\mathcal{F}_{\mathsf{NIZK}}$ that guarantees semi-malicious behavior in the latter protocol has been replaced by an invocation of $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ that simply allows the discrete logarithm of B to be extracted. Consequently, we omit any further formal description or security proof.

Realizing $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$. To realize $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, we requite a reactive generic MPC protocol that is secure with publicly verifiable identifiable abort against adversaries corrupting up to $n-1$ active participants. Due to the structure of our main protocol, which ejects parties from committees dynamically, it cannot be determined in advance which sets of parties will invoke $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, and consequently any "offline" preprocessing must actually be done "online" and with the same guarantees as the rest of the protocol. Thus preprocessing cost counts against the efficiency of the protocol realizing $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$. Also for the sake of efficiency, we prefer our protocol to natively compute arithmetic circuits over prime-order fields. To our knowledge, four recently-developed protocols satisfy or nearly satisfy our requirements. The first three derive from SPDZ [DPSZ12], and therefore natively compute arithmetic circuits with a round-count proportionate to circuit depth.

The protocol of Spini and Fehr [SF16] has low overhead relative to the original SPDZ protocol in the case that there are no cheaters, but it does not explicitly address public verifiability, does not address the necessary preprocessing phase, and achieves agreement with respect to the identity of cheaters only with "significant overhead."

The protocol of Cunningham et al. [CFY17] achieves public verifiability (which they refer to as *auditability*) and is capable of identifying *all* cheaters in an instance, rather than just one as per the standard definition of identifiable abort.[8] They also describe a preprocessing protocol with publicly verifiable

[8] Prior works [IOZ14, CL17] focused on feasibility results, in which context there is no

identifiable abort. The authors do not provide a cost analysis, but it seems likely that their protocol is computationally expensive, since it uses Somewhat Homomorphic Encryption (SHE) with NIZK proofs over the homomorphic operations extensively in the preprocessing phase. In our context, the number of such SHE/NIZK operations that must be verified by every party (active or passive) in the preprocessing phase grows with $\Omega(n^2 \cdot d)$, and they must be both computed and verified "online." Furthermore, this protocol is secure only when computing over fields where the order is a Sophie-Germain prime. This makes it incompatible with the order-fields of some elliptic curves, including the BLS12-381 curve used by prior works on SRS sampling.

The protocol of Baum et al. [BOS16] achieves the standard notion of publicly verifiable identifiable abort. They describe a preprocessing protocol with a matching security guarantee that is based on SHE and NIZK proofs, similarly to the protocol of Cunningham et al. [CFY17]. The authors do provide a cost analysis, which suggests that the number of NIZK proofs (over SHE ciphertexts) to be verified by each party (active or passive) grows with $\Omega(n^4 \cdot d)$.

A more-recent protocol of Baum et al. [BOSS20] achieves publicly verifiable identifiable abort so long as there is at least one honest party, which is sufficient for our purposes. It also requires only a constant number of rounds, regardless of circuit-depth, and it avoids the use of homomorphic encryption entirely. However, it computes over Boolean circuits, and rewriting our arithmetic circuit as a Boolean circuit yields tens or hundreds of billions of gates for the parameter regimes we expect in practice.

We think it likely that future work will improve upon these results. Thus, for the purpose of our efficiency analysis, we leave our costs in terms of circuit topology rather than choosing among the above protocols.

Realizing $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$. The $\mathcal{R}_{\mathsf{DL}}$ relation can be proven in zero knowledge via the Schnorr protocol [Sch89], and the proof can be made noninteractive and straight-line extractable by applying the transforms of Pass [Pas03] or Fischlin [Fis05], in order to UC-realize $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ in the $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$-hybrid random-oracle model. The prior work of Kohlweiss et al. [KMSV21] proved security only in the Algebraic Group Model (AGM) [FKL18]; in this model, the Schnorr protocol has a non-rewinding extractor by assumption. However, the AGM is incompatible with the UC Model, and so we assume the Fischlin transform is used for the sake of cost estimates.

5.2 Cost Analysis

In this section, we will analyze the costs of our protocol, and in particular its latency, *without* choosing a particular protocol to realize $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$. In service of this goal, we will introduce latency metrics for each major cost component. The computational and communication latency due to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ is determined

difference between identifying a single cheater or a larger set. The difference is relevent only when efficiency is a consideration.

by the number of gates, the circuit depth, and the number of parties involved in each invocation. The communication latency excluding $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ is determined by the total amount of data to be broadcasted, and the number of broadcast rounds. This leaves only the computational latency due to elliptic curve (EC) operations.

To account for the latency due to EC operations, we define two metrics. Our protocol can be broken down cleanly into individual broadcast rounds; in each round, every party begins by receiving the messages broadcasted in the previous round, whereupon it performs some computation, and then ends the round by broadcasting a new message. The *latency due to pairings* is the sum over all broadcast rounds of the maximum over the set of parties of the number of EC pairing operations performed by a given party in each round. Similarly, the *latency due to scalar multiplications* is the sum over all broadcast rounds of the maximum over the set of parties of the number of EC scalar multiplications performed by a given party in each round. For example, in state-of-the-art round-robin SRS-sampling protocols, each of the n parties must use $O(d)$ scalar multiplications and $O(1)$ pairing operations to verify the intermediate output of its predecessor;[9] since they are required to work sequentially, the pairing latency is $O(n)$ and the scalar latency is $O(d \cdot n)$.

Analysis of $\pi_{\mathsf{ExtSRS}}(n, \cdot, \cdot, \mathbb{G}_1, \mathbb{G}_2)$. The costs of this protocol differ depending on whether or not the cheater-identification phase occurs; thus we will give both optimistic and pessimistic costs. The generic component of π_{ExtSRS} comprises exactly $d - 1$ multiplication gates, $(d + 1) \cdot n$ input gates, and d output gates (optimistically) or $(d + 1) \cdot (n + 1) - 1$ output gates (pessimistically) arranged in a circuit of depth $\log_2 d + 3$ or $\log_2 d + 4$ (optimistically or pessimistically, respectively), and evaluated among n parties.

In the optimistic case, active participants must each compute $4d + 2$ EC scalar multiplications and 2 pairings, and they must each commit and reveal d elements of $\mathbb{G}_1$ using $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, and they must commit and prove one element of $\mathbb{G}_2$ using $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$, and they must each broadcast d elements of $\mathbb{G}_1$, plus one element of $\mathbb{G}_2$, and one bit. These operations are distributed over three broadcast rounds, assuming that the commitment and proof phases of $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ require one broadcast round each. Verifiers broadcast nothing, and need only evaluate 2 EC scalar multiplications and no pairings.

In the pessimistic case, at most one extra broadcast round is required (in addition to the extra circuit layer require by the generic component), during which one active participant is required to send at most $\log_2 d + \log_2 n$ bits. The same single active participant must evaluate at most $d + n$ additional EC scalar multiplications, and every *other* active participant and observing verifier must evaluate at most one additional EC scalar multiplication.

For the sake of concrete cost analysis, we will assume that the commitment

[9] Assuming such protocols use the same randomized well-formedness check as we do. This check seemingly does not appear in the prior academic literature on the subject, but it has been used in practice [Bow18].

phase of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ is realized via a single call to a random oracle, and a broadcast of the oracle's 2λ-bit output, and the decommitment phase is realized via the broadcast of the committed data. As we have previously said, we assume that $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ is realized via the Schnorr protocol under the Fischlin transform, in the $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$-hybrid random-oracle model. We take the cost of realizing $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ to be equivalent to $\lambda/\log_2\lambda$ repetitions of the underlying Schnorr protocol, plus one commitment.[10]

Summing it all up, we come to the following metrics: excluding generic costs (which were stated above), π_{ExtSRS} incurs a communication cost of at least three sequential broadcast rounds, over the course of which at least $n \cdot (2d \cdot |G_1| + 2|G_2| + 4\lambda + (|G_2| + |q|) \cdot \lambda/\log_2\lambda)$ bits must be broadcasted in total over all parties. At most, it incurs an additional communication cost of one broadcast round and $\log_2 d + \log_2 n$ broadcasted bits. For active participants, the scalar latency of the protocol is at least $4d + 2 + (2n-1) \cdot \lambda/\log_2\lambda$ and at most $5d + n + 2 + (2n-1) \cdot \lambda/\log_2\lambda$, while the pairing latency is always exactly 2. For verifiers, the scalar latency of the protocol is at least 2 and at most 3, while the pairing latency is 0.

Analysis of $\pi_{\mathsf{BilinearSRS}}(n, m, \cdot, \cdot, \mathbb{G}_1, \mathbb{G}_2)$. This protocol too has a variable cost that depends upon adversarial behavior, and so we will both compute the cost in the case of honest behavior, and bound the worst-case cost in the case of malicious behavior. We will make two major simplifications for the sake of analysis: first, we will assume that n/m is an integer, and thus all committees are the same size. Second, we will say for the sake of analysis that in every iteration of the main sampling loop, either an entire committee is removed, or one party is removed from each committee. Formally, we replace committee update instruction in Step 5 of $\pi_{\mathsf{BilinearSRS}}$ with

$$\mathbf{C}_{k+1,i,*} := \begin{cases} \mathbf{C}_{k,i,*} & \text{if } i \neq \ell \\ \varnothing & \text{if } i = \ell \end{cases}$$

for every $i \in [m]$. This makes the analysis simpler, and does not affect the asymptotics of our protocol, but does degrade its performance by a small constant for certain non-worst-case adversarial behavior patterns.

In each iteration of the main sampling loop, each committee invokes one instance of $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ as active participants, and also acts as passive verifiers for the other committee's instances of $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ (which are invoked concurrently). Verifiers for $\pi_{\mathsf{BilinearSRS}}$ passively verify all instances of $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and everyone (participants and verifiers alike) must compute $2d$ EC scalar multiplications and 2 pairing operations per iteration of the main sampling loop.

[10] In actuality, the Fischlin transform requires $c \cdot \lambda/\log_2\lambda$ repetitions of the Schnorr protocol's first message to be computed and broadcasted for some parameter c. The prover must also silently compute (but not broadcast) an asymptotically greater number of repetitions of the second and third messages, but these involve no EC scalar multiplications and thus do not impact the cost metrics we have devised.

In the case that all parties behave honestly, the best-case cost of π_{ExtSRS} is always achieved, and in each iteration of the main sampling loop, one committee goes from actively participating to passively verifying, but no individual parties are ever removed from any committee. Thus the total number of instances of π_{ExtSRS} invoked is $\sum_{i\in[m]} i = (m^2+m)/2$. This implies that the total numbers of generic input gates, multiplication gates, and output gates evaluated are $(d-1)\cdot(m^2+m)/2$, $(d+1)\cdot n\cdot(m+1)/2$, and $d\cdot(m^2+m)/2$ respectively, and all such gates are evaluated among exactly n/m parties. It also implies that total number of bits transmitted *excluding* the foregoing generic costs is $(m+1)\cdot n\cdot(2d\cdot|G_1|+2|G_2|+4\lambda+(|G_2|+|q|)\cdot\lambda/\log_2\lambda)/2$ in the case that all parties behave honestly.

Again in the case that all parties behave honestly, the number of iterations of the main sampling loop is m. This implies that the total circuit depth incurred is $3m+m\cdot\log_2 d$, the number of broadcast rounds not due to generic MPC is $3m$. In the i^{th} iteration of the main sampling loop, every remaining party participates in one instance of π_{ExtSRS} actively, and passively verifies $m-i$ instances of π_{ExtSRS}: thus for passive verifiers the EC pairing latency is $2m$ and the EC scalar latency is $m^2+2d\cdot m+m$, and for active participants, the EC pairing latency is $4m$ and the EC scalar latency is

$$\begin{aligned}&\sum_{i\in[m]} 2(m-i)+6d+2+(2n/m-1)\cdot\lambda/\log_2\lambda\\&=m^2+(6d+1)\cdot m+(2n-m)\cdot\lambda/\log_2\lambda\end{aligned}$$

In the case of dishonest behavior, we have as an invariant that in each iteration of the main loop, either one committee is removed, or one party is removed from each committee. This gives us an upper bound of $m+n/m$ iterations. The number of committees per iteration is always bounded above by m and the number of parties per committee by n/m. In order to simplify our calculations, we will assume these quantities apply in *all* iterations of the main loop, even though this strictly (and substantially) inflates our costs relative to the true worst-case protocol cost.

Plugging in the worst-case costs for π_{ExtSRS}, under the above simplifications, the total numbers of generic input gates, multiplication gates, and output gates required by our protocol in the worst case are strictly less than $(d+1)\cdot(n^2/m+n\cdot m)$, $(d-1)\cdot(m^2+n)$, and $(d+1)\cdot(n^2/m+n\cdot m)+d\cdot(m^2+n)$ respectively, arranged in circuits of total depth strictly no more than $(\log_2 d+4)\cdot(m+n/m)$ and evaluated among at most n/m parties. The total number of bits transmitted *excluding* the foregoing generic costs is strictly less than

$$\begin{aligned}&(n^2/m+n\cdot m)\cdot(2d\cdot|G_1|+2|G_2|+4\lambda+(|G_2|+|q|)\cdot\lambda/\log_2\lambda)\\&+(\log_2 d+\log_2(n/m))\cdot(m^2+n)\end{aligned}$$

Similarly, the total number of broadcast rounds required in the worst case (excluding those due to generic computation) is strictly no greater than $4(m+n/m)$. The EC pairing latency in the worst case is strictly no greater than $4(m+$

n/m) for active participants and $2(m+n/m)$ for passive verifiers. The EC scalar latency in the worst case is strictly no greater than $3(m^2+n)+2d\cdot(m+n/m)$ for passive verifiers, and for active participants it is strictly no greater than

$$3(m^2+n-m-n/m)+(7d+n/m+2+(2n/m-1)\cdot\lambda/\log_2\lambda)\cdot(m+n/m)$$

Finally, we will let $m=\sqrt{n}$ and consider the worst-case asymptotics implied by the above strict bounds. This gives us total generic input and output gates in $O(d\cdot n^{1.5})$, and total multiplication gates in $O(d\cdot n)$, all to be evaluated among $O(\sqrt{n})$ parties. The total number of bits broadcasted excluding generic computation is in $O(n^{1.5}\cdot d\cdot\lambda+n^{1.5}\cdot\lambda^2/\log\lambda)$. For active participants the EC scalar latency is in $O(d\cdot\sqrt{n}+n\cdot\lambda/\log\lambda)$, while for passive verifiers the EC scalar latency is in $O(n+d\cdot\sqrt{n})$, and for both active participants and passive verifiers, the pairing latency is in $O(\sqrt{n})$. Finally, if we assume for the sake of asymptotic analysis that the number of rounds required to instantiate generic MPC is proportionate to the depth of the circuit to compute, then the round count of the entire computation is bounded by $O(\sqrt{n}\log d)$.

5.3 Proof of Security

We proceed to prove the security of $\pi_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2)$.

Lemma 5.5. *Suppose $\mathbb{G}_1$ and $\mathbb{G}_2$ are a pair of groups, both of prime order q, such that there exists an efficient bilinear map $e:\mathbb{G}_1\times\mathbb{G}_2\to\mathbb{G}_\mathsf{T}$ and such that q is exponential in λ. For any $n,d\in O(\mathrm{poly}(\lambda))$, $\mathbf{X}\in\mathbb{G}_1^d$, and $Y\in\mathbb{G}_2$, with the constraints that $d\geq 2$ and there exists some $x\in\mathbb{Z}_q$ such that $\mathbf{Y}=x\cdot G_2$ and $\mathbf{X}_i=x^i\cdot G_1$ for every $i\in[d]$, it holds that $\pi_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2)$ statistically UC-realizes $[\![\mathcal{F}_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2)]\!]_{\mathsf{PV}}$ in the $([\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}(n,\mathbb{Z}_q)]\!]_{\mathsf{PV}},[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}},[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}})$-hybrid model.*

Proof. We begin by specifying a simulator $\mathcal{S}_{\mathsf{ExtSRS}}$, after which we will use a series of hybrid experiments to argue that

$$\begin{aligned}&\forall\mathcal{A}\ \forall\mathcal{Z},\\&\quad\left\{\mathrm{REAL}_{\pi_{\mathsf{ExtSRS}}(n,\{x^i\cdot G_1\}_{i\in[d]},x\cdot G_2,\mathbb{G}_1,\mathbb{G}_2),\mathcal{A},\mathcal{Z}}(\lambda)\right\}_{\lambda,n,d\in\mathbb{N}^+,x\in\mathbb{Z}_q}\\&\approx_{\mathrm{s}}\left\{\begin{array}{r}\mathrm{IDEAL}_{[\![\mathcal{F}_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2)]\!]_{\mathsf{PV}},\mathcal{S}^{\mathcal{A}}_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2),\mathcal{Z}}(\lambda):\\ \mathbf{X}\coloneqq\left\{x^i\cdot G_1\right\}_{i\in[d]},Y\coloneqq x\cdot G_2\end{array}\right\}_{\lambda,n,d\in\mathbb{N}^+,x\in\mathbb{Z}_q}\end{aligned}\tag{7}$$

Protocol 5.6. $\mathcal{S}^{\mathcal{A}}_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2)$**. SRS Extension**

This simulator is parameterized by the number of actively participating parties, $n\in\mathbb{N}^+$. It is also parameterized by a pair of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, both of the same prime order q, such that there exists a third group $\mathbb{G}_\mathsf{T}$ of order q and an efficient bilinear map $e:\mathbb{G}_1\times\mathbb{G}_2\to\mathbb{G}_\mathsf{T}$. Let G_1 and G_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$. In addition, it is parameterized by a vector of group elements $\mathbf{X}\in\mathbb{G}_1^d$ for some $d\in\mathbb{N}^+$ and a single group

element $Y \in \mathbb{G}_2$ with the constraint that there exists some $x \in \mathbb{Z}_q$ such that $\mathbf{Y} = x \cdot G_2$ and $\mathbf{X}_i = x^i \cdot G_1$ for every $i \in [d]$. This simulator interacts directly with the functionality $[\![\mathcal{F}_{\mathsf{ExtSRS}}(n, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$, and it has black-box access to an adversary $\mathcal{A}$ that statically corrupts the parties indexed by $\mathbf{P}^* \subseteq [n]$; it simulates an instance of the real-world UC experiment for $\pi_{\mathsf{ExtSRS}}(n, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)$ to $\mathcal{A}$, interacting with $\mathcal{A}$ on behalf of any honest parties and on behalf of the ideal oracles $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ and $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_{\mathsf{DL}}}$, and forwarding messages from the environment $\mathcal{Z}$ in its own ideal-world experiment to $\mathcal{A}$ and vice-versa.

Extension Sampling:

1. Upon receiving the initial batch of messages on behalf of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ from every party $\mathcal{P}_i$ for $i \in \mathbf{P}^*$, as specified by Step 1a of π_{ExtSRS}, store $(\mathtt{inputs}), \mathsf{sid}, i, \boldsymbol{\tau}_i, \boldsymbol{\mu}_{i,*})$ in memory and send $(\mathtt{sample}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_i$. Because our model is synchronous, this step should occur in parallel for all corrupt parties.

2. If $(\mathtt{candidate\text{-}output}, \mathsf{sid}, (\mathbf{A}, B))$ is received from $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ before all parties indexed by $\mathbf{P}^*$ have sent their initial batch of messages to $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ (as specified by Step 1a of π_{ExtSRS}), choose $c \in \mathbf{P}^*$ to be the smallest index corresponding to a party that has not followed the protocol, send $(\mathtt{abort}, \mathsf{sid})$ and then $(\mathtt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and halt. Otherwise, wait to receive $(\mathtt{candidate\text{-}output}, \mathsf{sid}, (\mathbf{A}, B))$ from $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ if there are any honest parties, or if there are no honest parties, then continue immediately.

3. Perform the interaction on behalf of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ that the corrupt parties would observe, were $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ also being activated by honest parties (if there are any) as per Step 1 of π_{ExtSRS}. If there are any honest parties present, then sample $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_q^d$ uniformly, and if all active participants are corrupt, then compute

$$\boldsymbol{\alpha}_j := \Big(\sum_{i \in [n]} \boldsymbol{\tau}_i \Big)^j + \sum_{i \in [n]} \boldsymbol{\mu}_{i,j}$$

   as $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ would in Step 1d of π_{ExtSRS}. Regardless, if at any point some corrupt party $\mathcal{P}_c$ fails to send an expected message to $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, then send $(\mathtt{abort}, \mathsf{sid})$ and then $(\mathtt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and halt.

4. Receive $(\mathtt{commit}, \mathsf{GenSID}(\mathsf{sid}, i), \mathbf{M}_{i,*})$ from every party $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, and $(\mathtt{commit}, \mathsf{GenSID}(\mathsf{sid}, i), \boldsymbol{\mu}'_i)$ on behalf of $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$. If any party $\mathcal{P}_c$ fails to send either of these messages, send $(\mathtt{abort}, \mathsf{sid})$ and then $(\mathtt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and halt. If there are honest parties participating, then send notifications to the corrupt

parties on behalf of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ as though the honest parties had committed per the instructions in Step 2 of π_{ExtSRS}. If there are no honest parties then ignore the `candidate-output` message that was previously received from $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and instead compute

$$\mathbf{A} := \left\{ \boldsymbol{\alpha}_j \cdot \mathbf{X}_j + \sum_{i \in [n]} \mathbf{M}_{i,j} \right\}_{j \in [d]} \quad \text{and} \quad b := \boldsymbol{\alpha}_1 + \sum_{i \in [n]} \boldsymbol{\mu}'_i$$

and $B := b \cdot Y$.

5. Receive $(\mathtt{open}, \mathsf{GenSID}(\mathsf{sid}, i))$ from every party $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, and $(\mathtt{prove}, \mathsf{GenSID}(\mathsf{sid}, i), (\mathbf{N}_i, Y))$ on behalf of $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$. If any party $\mathcal{P}_c$ fails to send either of these messages, or if for any c it holds that $\mathbf{N}_c \neq \boldsymbol{\mu}_{c,1} \cdot Y$, then send $(\mathtt{abort}, \mathsf{sid})$ and then $(\mathtt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and halt. Otherwise, if there are any honest parties participating, then choose $h \in [n]$ and $\mathbf{P}^{\mathsf{H}} \subset [n]$ arbitrarily such that $h \notin \mathbf{P}^{\mathsf{H}}$ and $\mathbf{P}^{\mathsf{H}} \cup \{h\} = [n] \setminus \mathbf{P}^*$, and then sample

$$\{(\boldsymbol{\mu}_{i,*})\}_{i \in \mathbf{P}^{\mathsf{H}}} \leftarrow \mathbb{Z}_q^{(d+1) \times |\mathbf{P}^{\mathsf{H}}|}$$

uniformly and compute

$$\begin{aligned}
\{\mathbf{M}_{i,*}\}_{i \in \mathbf{P}^{\mathsf{H}}} &:= \left\{ \left\{ (q - \boldsymbol{\mu}_{i,j}) \cdot \mathbf{X}_i \right\}_{j \in [d]} \right\}_{i \in \mathbf{P}^{\mathsf{H}}} \\
\mathbf{M}_{h,*} &:= \left\{ \left(\boldsymbol{\alpha}_j - \sum_{i \in \mathbf{P}^*} \boldsymbol{\mu}_{i,j} \right) \cdot \mathbf{X}_j + \sum_{i' \in \mathbf{P}^{\mathsf{H}}} \mathbf{M}_{i',j} - \mathbf{A}_j \right\}_{j \in [d]} \\
\{\mathbf{N}_i\}_{i \in \mathbf{P}^{\mathsf{H}}} &:= \left\{ (q - \boldsymbol{\mu}_{i,1}) \cdot Y \right\}_{i \in \mathbf{P}^{\mathsf{H}}} \\
\mathbf{N}_h &:= \left(\boldsymbol{\alpha}_1 - \sum_{i \in \mathbf{P}^*} \boldsymbol{\mu}_{i,1} \right) \cdot Y + \sum_{i' \in \mathbf{P}^{\mathsf{H}}} \mathbf{M}_{i',1} - B
\end{aligned}$$

and then for every $i \in [n] \setminus \mathbf{P}^*$ broadcast $\mathbf{M}_{i,*}$ on behalf of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and $\mathbf{N}_i$ on behalf of $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ to the corrupt parties as the openings to $\mathcal{P}_i$'s commitments.

6. Check whether

$$\boldsymbol{\alpha}_j \cdot \mathbf{X}_j + \sum_{i \in [n]} \mathbf{M}_{i,j} = \mathbf{A}_j \qquad \text{and} \qquad \boldsymbol{\alpha}_1 \cdot Y + \sum_{i \in [n]} \mathbf{N}_i = B$$

for all $j \in [d]$, and if there are any honest parties, then for the following set of conditions, consider their certified output values to be $(\mathbf{A}, B)$ if this relation holds, or $\perp$ if it does not.

(a) If there is at least one honest participant, and all active participants certified the output $(\mathbf{A}, B)$, then send $(\texttt{proceed}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ and halt.

(b) If all active participants are corrupt and they all certified some consistent non-$\perp$ output $(\mathbf{A}', B)$, then send $(\texttt{override}, \mathsf{sid}, (\mathbf{A}', b))$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ and halt.

(c) If there were two parties who certified two distinct non-$\perp$ outputs $(\mathbf{A}', B)$ and $(\mathbf{A}'', B)$, then let c' and c'' be their indices, and let them be the lowest such indices if more than one pair of parties meets this condition. Send $(\texttt{abort}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$. Find the minimal value of $j \in [d]$ such that $\mathbf{A}'_j \neq \mathbf{A}''_j$, and send $(\texttt{stooge}, \mathsf{sid}, c')$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ if $\mathbf{A}'_j \neq \mathbf{A}_j$, or else send $(\texttt{stooge}, \mathsf{sid}, c'')$ if $\mathbf{A}'_j = \mathbf{A}_j$. Halt regardless.

(d) Otherwise, it must be the case that at least one party certified $\perp$ and all other parties certified identical outputs. In this case, send $(\texttt{abort}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ and continue to Step 7 in the cheater-identification phase.

Cheater Identification:

7. If any party $\mathcal{P}_c$ fails to send the correct batch of `output` instructions to $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, then send $(\texttt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ and halt. On receiving $(\texttt{secret}, \mathsf{sid}, \tau)$ from $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, compute

$$\boldsymbol{\mu}_{h,*} := \left\{ \tau^j - \boldsymbol{\alpha}_j + \sum_{i \in [n] \setminus \{h\}} \boldsymbol{\mu}_{i,j} \right\}_{j \in [d]}$$

and sample $\{\boldsymbol{\tau}_i\}_{i \in \mathbf{P}^{\mathsf{H}}}$ uniformly subject to

$$\sum_{i \in [n]} \boldsymbol{\tau}_i = \tau$$

and output the elements of $\boldsymbol{\mu}$ and $\boldsymbol{\tau}$ to the corrupt parties on behalf of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ in the appropriate way, or else send $(\texttt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ if $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ would abort and identify $\mathcal{P}_c$ as a cheater.[a]

8. If the cheater-identification phase was initiated because the check in Step 6 failed, and there is at least one honest party, and all corrupt parties with lower indices than any honest parties broadcasted `ok`, then find the smallest values of $j \in [d]$ and $c \in \mathbf{P}^*$ (in that order) such that $\mathbf{M}_{c,j} \neq (q - \boldsymbol{\mu}_{c,j}) \cdot \mathbf{X}_j$, or if no such j exists, then the smallest value of $c \in \mathbf{P}^*$ such that $\mathbf{N}_c \neq (q - \boldsymbol{\mu}_{c,1}) \cdot Y$, and send $(\texttt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and then halt.

9. If the cheater-identification phase was initiated because the check in Step 6 failed, and there was a corrupt party $\mathcal{P}_{i'}$ who broadcasted `not-ok`, and whose index i' is smaller than the indices of all other parties who broadcasted `not-ok`, then wait to receive $(\texttt{malformed-output}, \mathsf{sid}, c, j)$ from $\mathcal{P}_{i'}$. If $\mathbf{M}_{c,j} \neq (q - \boldsymbol{\mu}_{c,j}) \cdot \mathbf{X}_j$ or $j = 0$ and $\mathbf{N}_c \neq (q - \boldsymbol{\mu}_{c,1}) \cdot Y$, then send $(\texttt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$. If $\mathcal{P}_{i'}$ fails to send a `malformed-output` message, or neither of the previous conditions hold, then send $(\texttt{stooge}, \mathsf{sid}, i')$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$. Halt regardless.

10. If the cheater-identification phase was initiated because one or more corrupt parties broadcasted `not-ok`, even though the check in Step 6 passed, then send $(\texttt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, where c is the index of the lowest-indexed corrupt party who broadcasted `not-ok`, and halt.

[a] When we say output an element in the appropriate way, we mean to mimic the behavior described in the Output phase of $\mathcal{F}_{\mathsf{MPC\text{-}IA}}$, including a `candidate-output` or `adv-output` message to $\mathcal{S}$.

We proceed to define a series of hybrid experiments. We begin with the real experiment,

$$\mathcal{H}_0 := \left\{ \mathrm{REAL}_{\pi_{\mathsf{ExtSRS}}(n, \{x^i \cdot G_1\}_{i \in [d]}, x \cdot G_2, \mathbb{G}_1, \mathbb{G}_2), \mathcal{A}, \mathcal{Z}}(\lambda) \right\}_{\lambda, n, d \in \mathbb{N}^+, x \in \mathbb{Z}_q}$$

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$, except that $\mathcal{Z}$ now communicates with a single, monolithic entity, $\mathcal{S}$, which internally emulates an instance of the real-world experiment for $\mathcal{A}$ (to which $\mathcal{S}$ has black-box access), in which $\mathcal{S}$ itself plays the roles of all honest parties and oracles (excluding $\mathcal{Z}$ and $\mathcal{A}$), following their code exactly as specified in π_{ExtSRS}, and forwarding all messages between the emulated experiment's environment and $\mathcal{Z}$. $\mathcal{H}_1$ differs from $\mathcal{H}_0$ only syntactically; the two have identical distributions.

Hybrid $\mathcal{H}_2$. This hybrid is identical to $\mathcal{H}_1$ when all parties are corrupt. If there is at least one honest party, then the method by which the values in the experiment are calculated is altered. Let $\mathcal{P}_h$ be an (arbitrary) honest party. Rather than sampling a uniform value for $\boldsymbol{\tau}_h$ and $\boldsymbol{\mu}_{h,*}$ and using these to compute $\boldsymbol{\alpha}$ in the way $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ would, $\mathcal{S}$ instead samples $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_q^d$ and $\tau \leftarrow \mathbb{Z}_q$ uniformly, and then computes $(\mathbf{A}, B)$ from τ in the way that $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ would. Then, using $(\mathbf{A}, B)$, $\boldsymbol{\alpha}$, and $\{\boldsymbol{\mu}_{i,*}\}_{i \in [n] \setminus \{h\}}$, $\mathcal{S}$ computes

$$\begin{aligned} \mathbf{M}_{h,*} &:= \left\{ \left(\boldsymbol{\alpha}_j - \sum_{i \in \mathbf{P}^*} \boldsymbol{\mu}_{i,j} \right) \cdot \mathbf{X}_j + \sum_{i' \in \mathbf{P}^{\mathsf{H}}} \mathbf{M}_{i',j} - \mathbf{A}_j \right\}_{j \in [d]} \\ \mathbf{N}_h &:= \left(\boldsymbol{\alpha}_1 - \sum_{i \in \mathbf{P}^*} \boldsymbol{\mu}_{i,1} \right) \cdot Y + \sum_{i' \in \mathbf{P}^{\mathsf{H}}} \mathbf{M}_{i',1} - B \end{aligned} \tag{8}$$

and uses these values. Only if the simulated protocol enters the cheater-identification phase (Step 6 of π_{ExtSRS}) will $\mathcal{S}$ need to calculate and reveal $\boldsymbol{\tau}_h$

and $\boldsymbol{\mu}_{h,*}$, which it has not otherwise needed. In this case, it calculates

$$\begin{aligned} \boldsymbol{\mu}_{h,*} &:= \left\{ \tau^j - \boldsymbol{\alpha}_j + \sum_{i \in [n] \setminus \{h\}} \boldsymbol{\mu}_{i,j} \right\}_{j \in [d]} \\ \boldsymbol{\tau}_h &:= \tau - \sum_{i \in [n] \setminus \{h\}} \boldsymbol{\tau}_i \end{aligned} \tag{9}$$

Although the sequence by which the values are calculated differs, $\mathcal{H}_2$ is distributed identically to $\mathcal{H}_1$. Notice that Equation Pair 8 is simply an algebraic rewriting of the equations in Steps 2 and 3 of π_{ExtSRS}, and likewise Equation Pair 9 is a rewriting of the equations evaluated by $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ in Steps 1a and 1d of π_{ExtSRS}. Because identical constraints relate all of the values in both $\mathcal{H}_2$ and $\mathcal{H}_1$, and because $\boldsymbol{\tau}_h$ and $\boldsymbol{\mu}_{h,*}$ were sampled uniformly in $\mathcal{H}_1$, whereas $\boldsymbol{\tau}$ and $\boldsymbol{\alpha}$ are chosen uniformly in $\mathcal{H}_2$, it must be the case that $\mathcal{H}_2 = \mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. This hybrid differs from $\mathcal{H}_2$ in only one respect: the randomized checks performed by all honest active participants in Step 4 of π_{ExtSRS} are replaced by deterministic checks. Specifically if there is at least one honest active participant, then whenever it would have run the original check, $\mathcal{S}$ instead checks whether

$$\begin{aligned} & e(G_1, B) = e(\mathbf{A}_1, G_2) \\ \wedge \quad & e(\mathbf{A}_{j-1}, B) = e(\mathbf{A}_j, G_2) \qquad \forall j \in [2, d] \end{aligned}$$

Observe that by the correctness of the bilinear mapping operator e, this check verifies that the SRS $(\mathbf{A}, B)$ is well-formed (i.e., it is in the image of $\mathsf{BilinearSRS}((G_1^d, G_2), \cdot)$) with perfect soundness. Thus, the adversary can distinguish $\mathcal{H}_3$ from $\mathcal{H}_2$ only when there is at least one honest active participant, by contriving generate an ill-formed output $(\mathbf{A}, B)$ which nevertheless passes the probabilistic check in Step 4 of π_{ExtSRS}.

Suppose for some j, we define Δ to be the additive deviation of $\mathbf{A}_j$ from its ideal value, where the ideal value is defined with respect to B. That is, if $b \in \mathbb{Z}_q$ is chosen such that $B = b \cdot G_2$, then let $\Delta = \mathbf{A}_j - b^j \cdot G_1$. On the left hand side of the probabilistic check, $\mathbf{A}_j$ is multiplied by $\mathbf{r}_j$, and on the right hand side, it is multiplied by $\mathbf{r}_{j+1}$; thus the two sides of the checking equation differ by $e((|\mathbf{r}_{j+1} \cdot b - \mathbf{r}_j|) \cdot \Delta, G_2)$. Over the uniform choice of $\mathbf{r}$, $\Pr[\mathbf{r}_{j+1} \cdot b = \mathbf{r}_j] = 1/q$. The adversary cannot improve its probability of success by offsetting additional elements of $\mathbf{A}$, since those elements are also multiplied by the corresponding uniformly chosen values in $\mathbf{r}$, and the probability that the intended offset occurs is also $1/q$. The probabilistic check is repeated once for each honest active participant, with independently sampled value values of $\mathbf{r}$, and the adversary distinguishes only if it passes *all* instances of the check. Its probability of doing so is upper-bounded by $1/q \in \mathsf{negl}(\lambda)$, and thus $\mathcal{H}_3 \approx_{\mathrm{s}} \mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. This hybrid is identical to $\mathcal{H}_3$ when all parties are corrupt. When there is at least one honest active participant, this hybrid differs from $\mathcal{H}_3$ in that

the deterministic well-formedness check that we introduced in $\mathcal{H}_3$ is replaced; instead $\mathcal{S}$ verifies that

$$\boldsymbol{\alpha}_j \cdot \mathbf{X}_j + \sum_{i \in [n]} \mathbf{M}_{i,j} = \mathbf{A}_j \qquad \text{and} \qquad \boldsymbol{\alpha}_1 \cdot Y + \sum_{i \in [n]} \mathbf{N}_i = B$$

for every $j \in [d]$. This check not only guarantees the well-formedness of $(\mathbf{A}, B)$ (again, in the sense that it is in the image of $\mathsf{BilinearSRS}((G_1^d, G_2), \cdot)$), but also guarantees with perfect soundness that an abort occurs if the honest parties' outputs to the environment are not actually equal to $(\mathbf{A}, B)$. Consequently, $\mathcal{H}_4$ and $\mathcal{H}_3$ can be distinguished if and only if the adversary can find values of $\mathbf{M}_{i,*}$ and $\mathbf{N}_i$ for $i \in \mathbf{P}^*$ such that if we set

$$\mathbf{A}' \coloneqq \left\{ \boldsymbol{\alpha}_j \cdot \mathbf{X}_j + \sum_{i \in [n]} \mathbf{M}_{i,j} \right\}_{j \in [d]} \qquad \text{and} \qquad B' \coloneqq \boldsymbol{\alpha}_1 \cdot Y + \sum_{i \in [n]} \mathbf{N}_i$$

then $\mathbf{A}' \neq \mathbf{A}$ or $B' \neq B$, but $(\mathbf{A}', B')$ is a well-formed SRS. In order to do this, the adversary must offset the value of $\mathbf{M}_{i,*}$ for at least one $i \in \mathbf{P}^*$ and every $j \in [d]$, relative to the values that $\mathcal{S}$ expects. However, it must commit to these values before $\mathbf{M}_{h,*}$ is revealed to it (and indeed, $\mathcal{S}$ need not even have flipped the coins required to compute $\mathbf{M}_{h,*}$ before the adversary commits itself). Consequently, the chance that it guesses a set of offsets that yields a well-formed SRS is upper-bounded by $1/q^{d-1}$, and if $d \geq 2$, then $\mathcal{H}_4 \approx_{\mathrm{s}} \mathcal{H}_3$.

Hybrid $\mathcal{H}_5$. This hybrid differs from $\mathcal{H}_4$ *only* when all active participants are corrupt. In this case, the code of all honest passive verifiers is removed from $\mathcal{S}$, and they are replaced by dummy parties that interact (in the typical way) with $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, which interacts with $\mathcal{S}$ in turn. $\mathcal{S}$ sends $(\texttt{sample}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ on behalf of each $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ when it receives the first message in the protocol from $\mathcal{P}_i$ on behalf of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$. If at any point, some party $\mathcal{P}_c$ omits an expected message (including the `prove` message for $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$, which must include a specific set of values in order for $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$ to produce output for the other parties), or sends a message that would cause $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ to abort, $\mathcal{S}$ triggers $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ to abort, identifying $\mathcal{P}_c$ as the cause.[11] Upon the completion of Step 2 of π_{ExtSRS} by the corrupt parties, $\mathcal{S}$ takes $\boldsymbol{\mu}_i'$ to be the value committed by each $\mathcal{P}_i$ to $[\![\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}]\!]_{\mathsf{PV}}$, computes

$$\mathbf{A} \coloneqq \left\{ \boldsymbol{\alpha}_j \cdot \mathbf{X}_j + \sum_{i \in [n]} \mathbf{M}_{i,j} \right\}_{j \in [d]} \quad \text{and} \quad b \coloneqq \boldsymbol{\alpha}_1 + \sum_{i \in [n]} \boldsymbol{\mu}_i'$$

. Finally, at the end of the protocol, all corrupt parties broadcast either `ok` or `not-ok` (in which case one party also broadcasts `malformed-output`), and $\mathcal{S}$ then runs single modified copy of the verifiers' code in Steps 9 though 13 of π_{ExtSRS}. Specifically, it is modified in the following ways:

[11] The simulator can do this by sending $(\texttt{abort}, \mathsf{sid})$, and $(\texttt{stooge}, \mathsf{sid}, c)$ in sequence, or only $(\texttt{stooge}, \mathsf{sid}, c)$ if an `abort` message has already been sent.

whenever $\mathcal{V}$ would output $(\texttt{abort}, \mathsf{sid}, c)$ to the environment, $\mathcal{S}$ instead sends $(\texttt{abort}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, followed by $(\texttt{stooge}, \mathsf{sid}, c)$, and whenever $\mathcal{V}$ would output $(\texttt{output}, \mathsf{sid}, (\mathbf{A}', B))$ to the environment, $\mathcal{S}$ instead sends $(\texttt{override}, \mathsf{sid}, (\mathbf{A}', b))$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$. The code by which $\mathcal{V}$ computes B in Step 11 of π_{ExtSRS} can be omitted, since $\mathcal{S}$ has already computed b such that $B = b \cdot Y$; it can be verified by inspection that $\mathcal{S}$ extracts the discrete logarithms of the values $\mathcal{V}$ uses to compute B, and computes b using the same equation. $\mathcal{S}$ halts when $\mathcal{V}$ would.

Because $\mathcal{H}_5$ differs from $\mathcal{H}_4$ only when all active participants are corrupt, and the distributions of all values sent by $\mathcal{S}$ to the corrupt participants are unchanged, relative to $\mathcal{H}_4$, the only way the two hybrids could be distinguished is via the outputs of honest passive verifiers; thus for the remainder of the argument concerning $\mathcal{H}_5$, we consider only aborts and outputs produced by such verifiers. We have already argued that the value of b passed to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ in $\mathcal{H}_5$ is the discrete logarithm with respect to Y of the value of B computed by $\mathcal{V}$ in $\mathcal{H}_4$, and by virtue of the fact that $\mathcal{S}$ runs a modified version of the code of $\mathcal{V}$, it is easy to see by inspection that the dummy verifiers in $\mathcal{H}_5$ receive an abort under exactly the same conditions that $\mathcal{V}$ aborts in $\mathcal{H}_4$, and the same party is identified as a cheater. Finally, we observe that if no abort occurs, $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ simply passes $\mathbf{A}'$ through to the dummy verifiers, and thus the output value in the case that no cheating occurs is identical in $\mathcal{H}_5$ and $\mathcal{H}_4$. Thus the two hybrids are distributed identically.

Hybrid $\mathcal{H}_6$. This hybrid is identical to $\mathcal{H}_5$ when all parties are corrupt, and differs when there is at least one honest active participant. The output-producing code of all honest active participants and all honest passive verifiers is deleted from $\mathcal{S}$; they are replaced in their interactions with the environment by dummy parties that interact with $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, which interacts with $\mathcal{S}$ in turn. $\mathcal{S}$ sends $(\texttt{sample}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ on behalf of each $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ when it receives the first message in the protocol from $\mathcal{P}_i$ on behalf of $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, and if $\mathcal{S}$ receives a `candidate-output` message from $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ before some corrupt participant $\mathcal{P}_i$ activates $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$, then it triggers $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ to abort, identifying $\mathcal{P}_i$ as the cheater.

In $\mathcal{H}_6$, when there is at least one honest active participant, $\mathcal{S}$ samples $\boldsymbol{\alpha}_i \leftarrow \mathbb{Z}_q$ only for $i \in \mathbf{P}^*$ rather than for $i \in [n]$ (as it did in $\mathcal{H}_5$), and it does not sample τ or compute $(\mathbf{A}, B)$, but instead receives $(\texttt{candidate-output}, \mathsf{sid}, (\mathbf{A}, B))$ from $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$. Using these values it computes $\mathbf{M}_{h,*}$ and $\mathbf{N}_h$ as before. At the end of the protocol, all corrupt parties certify their outputs (or $\perp$), and $\mathcal{S}$ determines whether an honest party *would* certify $(\mathbf{A}, B)$ or $\perp$, via the second deterministic well-formedness check previously introduced two hybrids previously. $\mathcal{S}$ then determines how to interact with $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ and complete the simulation via a modified version of the honest-party code in Step 5 though 8 of π_{ExtSRS}. Specifically, the code of the simulated honest parties is modified in the following ways: whenever the honest parties would output $(\texttt{output}, \mathsf{sid}, (\mathbf{A}, B))$ to the environment, $\mathcal{S}$ instead sends $(\texttt{proceed}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and whenever the honest parties would jump to Step 6 (which implies that they will eventually

abort), $\mathcal{S}$ instead sends $(\texttt{abort}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, and receives $(\texttt{secret}, \mathsf{sid}, \tau)$ in response, which enables it to compute values for $\boldsymbol{\tau}_h$ and $\boldsymbol{\mu}_{h,*}$ that are consistent with the transcript so far, using the same equations as in $\mathcal{H}_5$. It then uses these values to simulate Step 6 of π_{ExtSRS} to the corrupt participants, after which $\mathcal{S}$ sends $(\texttt{stooge}, \mathsf{sid}, c)$ to $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ whenever the honest parties would output $(\texttt{abort}, \mathsf{sid}, c)$ to the environment.

We note first of all that the protocol values in $\mathcal{H}_6$ are calculated via equations identical to those used to compute the values in $\mathcal{H}_5$ (though some parts of the calculation are now done by $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ instead of $\mathcal{S}$). It can be seen by inspection that the values of $\boldsymbol{\tau}_h$ and $\boldsymbol{\mu}_{h,*}$ used in simulating Step 6 of π_{ExtSRS} in $\mathcal{H}_6$ are distributed identically to the ones used in $\mathcal{H}_5$, and by virtue of the fact that $\mathcal{S}$ uses a modified version of the honest parties' code, it is easy to verify that the dummy honest participants in $\mathcal{H}_6$ receive an abort under exactly the same conditions as the honest parties aborted independently in $\mathcal{H}_5$, and that the same cheating party is identified to the functionality as an honest party would identify, and that the same output is produced if no abort occurs.

There remain two potential differences about which we must argue: in $\mathcal{H}_6$, $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ checks to make sure only corrupt parties are identified as cheaters, so we must argue that no honest party could be framed as a cheater in $\mathcal{H}_5$, and in $\mathcal{H}_6$, the dummy verifiers always receive the same output as the dummy honest parties from $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$, so we must argue that the verifiers and honest parties always computed the same output in $\mathcal{H}_5$, even though they ran different code. The first point can be verified by exhaustive inspection: honest parties send all messages that they're expected to send (by definition), never send values that are inconsistent (since the consistency criteria are defined relative to honest behavior), and never incorrectly blame other parties who have acted consistently; these are the only criteria under which any party is ever declared to have cheated. The second requires us only to note that the conditions under which outputs are produced in the honest parties' code, in Steps 5a, 5b, 5c, and 8, exactly match the conditions in 10, 11, 12, and 13 respectively. It follows that $\mathcal{H}_6$ and $\mathcal{H}_5$ are identically distributed.

Finally, we observe that apart from syntactic differences, $\mathcal{H}_6$ is identical to the ideal-world experiment. That is, in $\mathcal{H}_5$, $\mathcal{S}$ behaves exactly the same as $\mathcal{S}^{\mathcal{A}}_{\mathsf{ExtSRS}}$, and it follows that

$$\mathcal{H}_5 = \left\{ \begin{array}{r} \mathrm{IDEAL}_{[\![\mathcal{F}_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2)]\!]_{\mathsf{PV}},\mathcal{S}^{\mathcal{A}}_{\mathsf{ExtSRS}}(n,\mathbf{X},Y,\mathbb{G}_1,\mathbb{G}_2),\mathcal{Z}}(\lambda) : \\ \mathbf{X} \coloneqq \left\{x^i \cdot G_1\right\}_{i\in[d]}, Y \coloneqq x \cdot G_2 \end{array} \right\}_{\lambda,n,d\in\mathbb{N}^+,x\in\mathbb{Z}_q}$$

and, by transitivity, that the real and ideal-world experiments are statistically indistinguishable as stated in Equation 7. □

Theorem 5.7. *Suppose $\mathbb{G}_1$ and $\mathbb{G}_2$ are a pair of groups, both of prime order q, such that there exists an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$ and such that q is exponential in the statistical security parameter. For any $n, d, m \in O(\mathrm{poly}(\lambda))$, $\mathbf{X} \in \mathbb{G}_1^d$, and $Y \in \mathbb{G}_2$, with the constraints that $d \geq 2$,*

$1 \leq m \leq n$, *and there exists some* $x \in \mathbb{Z}_q$ *such that* $\mathbf{Y} = x \cdot G_2$ *and* $\mathbf{X}_i = x^i \cdot G_1$ *for every* $i \in [d]$, *it holds that* $\pi_{\mathsf{BilinearSRS}}(n, m, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)$ *statistically UC-realizes* $[\![\mathcal{F}_{\mathsf{PostTrans}}(\mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}, n, (\mathbf{X}, Y), m + n/m)]\!]_{\mathsf{PV}}$ *in the* $[\![\mathcal{F}_{\mathsf{ExtSRS}}(*, *, *, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$*-hybrid model.*

Proof Sketch. The form of a full proof of Theorem 5.7 would be similar to a proof of Theorem 3.9 (though without the elaborate reduction the security of the round-robin protocol), and thus we give a simplified sketch, noting mainly the differences. Since we are only sketching a proof, we will assume for simplicity that there is a single honest party. In each iteration of the primary loop of $\pi_{\mathsf{BilinearSRS}}$, $\mathcal{S}$ embeds a fresh candidate SRS provided by $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$ in the protocol via the `candidate-output` message of the instance of $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ in which the honest party as an active participant. If that same instance does not abort and its output becomes the finalized intermediate SRS for its iteration of the loop, then $\mathcal{S}$ does not reject, but begins accumulating the product of the finalized b values it extracts in its role as $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ in subsequent iterations of the main loop; otherwise, $\mathcal{S}$ sends `reject` and receives a fresh candidate from $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$. At the end of the protocol, let $\hat{b}$ be the accumulated product of the b values extracted in each iteration of the protocol's main loop after the functionality's candidate SRS is finalized. $\hat{b}$ is sent as the adversary's bias value in $\mathcal{S}$'s `accept` message to $[\![\mathcal{F}_{\mathsf{PostTrans}}]\!]_{\mathsf{PV}}$.

Note that unlike in the case of π_{Compiler}, the output of a particular instance of $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ is not finalized immediately simply because that instance is the lowest-indexed one that did not abort in a particular iteration of the loop: instead, it is verified using a randomized check in Step 4 of $\pi_{\mathsf{BilinearSRS}}$. If the check passes, the output is finalized, and if it fails, *no* output is finalized for that iteration. The latter case can only occur when all active participants in the instance of $[\![\mathcal{F}_{\mathsf{ExtSRS}}]\!]_{\mathsf{PV}}$ are corrupt. Proving that the ideal-world experiment with $\mathcal{S}$ constructed as described above is indistinguishable from the real-world experiment will involve a series of hybrid experiments: in each sequential hybrid, one instance of the randomized check will be replaced with an exact check that the to-be-finalized intermediate SRS is well-formed; this exact transition between hybrid-experiments is also used our proof of Lemma 5.5, and here, as there, the two are statistically indistinguishable. All other differences are purely syntactic, and so the real and ideal experiments are statistically indistinguishable, by transitivity. □

Corollary 5.8. *Suppose* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *are a pair of groups, both of prime order* q, *such that there exists an efficient bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$ *and such that* q *is exponential in the statistical security parameter. For any* $n, d, m \in O(\mathrm{poly}(\lambda))$, $\mathbf{X} \in \mathbb{G}_1^d$, *and* $Y \in \mathbb{G}_2$, *with the constraints that* $d \geq 2$, $1 \leq m \leq n$, *and there exists some* $x \in \mathbb{Z}_q$ *such that* $\mathbf{Y} = x \cdot G_2$ *and* $\mathbf{X}_i = x^i \cdot G_1$ *for every* $i \in [d]$, *it holds that* $\pi_{\mathsf{BilinearSRS}}(n, m, \mathbf{X}, Y, \mathbb{G}_1, \mathbb{G}_2)$ *statistically UC-realizes* $[\![\mathcal{F}_{\mathsf{PreTrans}}(\mathsf{BilinearSRS}_{\mathbb{G}_1,\mathbb{G}_2,d}, n, (\mathbf{X}, Y))]\!]_{\mathsf{PV}}$ *in the* $[\![\mathcal{F}_{\mathsf{ExtSRS}}(*, *, *, \mathbb{G}_1, \mathbb{G}_2)]\!]_{\mathsf{PV}}$*-hybrid model.*

Proof. By conjunction of Theorems 5.7 and 4.8, and the fact that BilinearSRS

is a homomorphic update function as discussed in Section 4.4. □

Acknowledgements

We thank Alon Rosen for a helpful discussion and Peter Scholl for answers to questions about his works on generic MPC with IA. We furthermore thank an anonymous reviewer for making us aware of the randomized well-formedness check described in Section 5, and for pushing us to optimize our "efficient" protocol. Ran Cohen's research is supported in part by NSF grant no. 2055568. The other authors are supported in part by NSF grants 1816028 and 1646671.

References

[ABC+85] Baruch Awerbuch, Manuel Blum, Benny Chor, Shafi Goldwasser, and Silvio Micali. How to implement Bracha's $O(\log n)$ Byzantine agreement algorithm, 1985. Unpublished manuscript.

[Abe99] Masayuki Abe. Mix-networks on permutation networks. In *Advances in Cryptology – ASIACRYPT 1999*, pages 258–273, 1999.

[ABMO15] Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. Complete characterization of fairness in secure two-party computation of Boolean functions. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part I*, pages 199–228, 2015.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology – EUROCRYPT 2012*, pages 483–501, 2012.

[ALR13] Gilad Asharov, Yehuda Lindell, and Tal Rabin. A full characterization of functions that imply fair coin tossing and ramifications to fairness. In *Proceedings of the 10th Theory of Cryptography Conference, TCC 2013*, pages 243–262, 2013.

[AO16] Bar Alon and Eran Omri. Almost-optimally fair multiparty coin-tossing with nearly three-quarters malicious. In *Proceedings of the 14th Theory of Cryptography Conference, TCC 2016-B, part I*, pages 307–335, 2016.

[Ash14] Gilad Asharov. Towards characterizing complete fairness in secure two-party computation. In *Proceedings of the 11th Theory of Cryptography Conference, TCC 2014*, pages 291–316, 2014.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology – EUROCRYPT 2004*, pages 56–73, 2004.

[BCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *Proceedings of the 36th IEEE Symposium on Security and Privacy, (S&P)*, pages 287–304, 2015.

[BDD+21] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In *Advances in Cryptology – EUROCRYPT 2021, part III*, pages 429–459, 2021.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *Proceedings of the 9th Conference on Security and Cryptography for Networks (SCN)*, pages 175–196, 2014.

[BF01] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *Journal of the ACM*, 48(4):702–722, 2001.

[BGG18] Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. In *Proceedings of the 22nd Financial Cryptography and Data Security (FC)*, pages 64–77, 2018.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *IACR Cryptol. ePrint Arch.*, 2017:1050, 2017.

[BHLT17] Niv Buchbinder, Iftach Haitner, Nissan Levi, and Eliad Tsfadia. Fair coin flipping: Tighter analysis and the many-party case. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2580–2600, 2017.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *Advances in Cryptology – EUROCRYPT 2009*, pages 1–35, 2009.

[BKRS18] Elette Boyle, Saleet Klein, Alon Rosen, and Gil Segev. Securing Abe's mix-net against malicious verifiers via witness indistinguishability. In *Proceedings of the 11th Conference on Security and Cryptography for Networks (SCN)*, pages 274–291, 2018.

[BLOO11] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. In *Advances in Cryptology – CRYPTO 2011*, pages 277–296, 2011.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 503–513, 1990.

[BOO15] Amos Beimel, Eran Omri, and Ilan Orlov. Protocols for multiparty coin toss with a dishonest majority. *Journal of Cryptology*, 28(3):551–600, 2015.

[BOS16] Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. In *Proceedings of the 14th Theory of Cryptography Conference, TCC 2016-B, part I*, pages 461–490, 2016.

[BOSS20] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In *Advances in Cryptology – CRYPTO 2020, part II*, pages 562–592, 2020.

[Bow18] Sean Bowe. https://github.com/ebfull/powersoftau/blob/54294159591750822 07fd61c10319e47a6b56e87/src/lib.rs#L633, 2018.

[Bra87] Gabriel Bracha. An O(log n) expected rounds randomized Byzantine generals protocol. *Journal of the ACM*, 34(4):910–920, 1987.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *Advances in Cryptology – CRYPTO 2014, part I*, pages 480–499, 2014.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In *Advances in Cryptology – CRYPTO 2020, part III*, pages 64–93, 2020.

[CDKs22] Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. Guaranteed output in $O(\sqrt{n})$ rounds for round-robin sampling protocols. In *Advances in Cryptology – EUROCRYPT 2022*, pages 241–271, 2022.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology – CRYPTO 2001*, pages 19–40, 2001.

[CFY17] Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In *Proceedings of the 10th International Conference on Information Theoretic Security (ICITS)*, pages 110–134, 2017.

[CHI+21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy, (S&P)*, pages 590–607, 2021.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020, part I*, pages 738–768, 2020.

[CHOR22] Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. From fairness to full security in multiparty computation. *Journal of Cryptology*, 35(1):4, 2022.

[CL17] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, 2017.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2002.

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

[CsW19] Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure MPC with sublinear communication complexity. In *Advances in Cryptology – CRYPTO 2019, part II*, pages 30–60, 2019.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Triply adaptive UC NIZK. *IACR Cryptol. ePrint Arch.*, 2020.

[Dac20] Dana Dachman-Soled. Revisiting fairness in MPC: polynomial number of parties and general adversarial structures. In *Proceedings of the 18th Theory of Cryptography Conference, TCC 2020, part II*, pages 595–620, 2020.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, pages 643–662, 2012.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Advances in Cryptology – CRYPTO 2005*, pages 152–168, 2005.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Advances in Cryptology – CRYPTO 2018, part II*, pages 33–62, 2018.

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *Advances in Cryptology – CRYPTO 2018, part II*, pages 331–361, 2018.

[FS01] Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In *Proceedings of the 4th International Conference on the Theory and Practice of Public-Key Cryptography (PKC)*, pages 300–316, 2001.

[GHKL08] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 413–422, 2008.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology – EUROCRYPT 1999*, pages 295–310, 1999.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of Pedersen's distributed key generation protocol. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA)*, pages 373–390, 2003.

[GK09] S. Dov Gordon and Jonathan Katz. Complete fairness in multiparty computation without an honest majority. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 19–35, 2009.

[GK12] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. *Journal of Cryptology*, 25(1):14–40, 2012.

[GKM+00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. The relationship between public key encryption and oblivious transfer. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–335, 2000.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Advances in Cryptology – CRYPTO 2018, part III*, pages 698–728, 2018.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography – VOLUME 2: Basic Applications*. Cambridge University Press, 2004.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM*, 59(3):11:1–11:35, 2012.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016, part II*, pages 305–326, 2016.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical non-interactive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[HMR+19] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32(2):265–323, 2019.

[HT17] Iftach Haitner and Eliad Tsfadia. An almost-optimally fair three-party coin-flipping protocol. *SIAM Journal on Computing*, 46(2):479–542, 2017.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology – CRYPTO 2014, part II*, pages 369–386, 2014.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591, 2008.

[KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In *Advances in Cryptology – ASIACRYPT 2021, part III*, pages 98–127, 2021.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Proceedings of the 10th Theory of Cryptography Conference, TCC 2013*, pages 477–498, 2013.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In *Advances in Cryptology – ASIACRYPT 2015, part I*, pages 52–75, 2015.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology – ASIACRYPT 2010*, pages 177–194, 2010.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, part II*, pages 705–734, 2016.

[LM20] Chen-Da Liu-Zhang and Ueli Maurer. Synchronous constructive cryptography. In *Proceedings of the 18th Theory of Cryptography Conference, TCC 2020, part II*, pages 439–472, 2020.

[Mak14] Nikolaos Makriyannis. On the classification of finite Boolean functions up to fairness. In *Proceedings of the 9th Conference on Security and Cryptography for Networks (SCN)*, pages 135–154, 2014.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *Proceedings of the 26th ACM Conference on Computer and Communications Security, (CCS)*, pages 2111–2128, 2019.

[MNS16] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. *Journal of Cryptology*, 29(3):491–513, 2016.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In *Advances in Cryptology – CRYPTO 2003*, pages 316–337, 2003.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology – EUROCRYPT 1991*, 1991.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology – CRYPTO 1989*, pages 239–252, 1989.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology – CRYPTO 2001*, pages 566–598, 2001.

[SF16] Gabriele Spini and Serge Fehr. Cheater detection in SPDZ multiparty computation. In *Proceedings of the 9th International Conference on Information Theoretic Security (ICITS)*, pages 151–176, 2016.

[Sta96] Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology – EUROCRYPT 1996*, pages 190–199, 1996.

[SV15] Berry Schoenmakers and Meilof Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In *Proceedings of the 13th International Conference on Applied Cryptography and Network Security (ACNS)*, pages 3–22, 2015.

[Wik05] Douglas Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *Advances in Cryptology – ASIACRYPT 2005*, pages 273–292, 2005.