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

Ran Cohen[1], Jack Doerner[2], Yashvanth Kondi[2], and abhi shelat[2]

[1] Reichman University
[2] Northeastern University

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.

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 who may corrupt up to $n-1$ parties, all honest parties always learn their outputs from the computation. This property, for example, is needed 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. However, the seminal result of Cleve [23] 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 [35,4,49,2,3]. 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

* The full version of this paper is available at https://eprint.iacr.org/2022/257

functions) [36,25,26]. 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 [36] 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 [34,33]) 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* [42,25], 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 $\Omega(n)$.[3]

The $1/p$ relaxation. A closer look at Cleve's attack [23] 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 state of the art for coin-tossing is the work of Buchbinder et al. [16] 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 [5,23] 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 $O(1/r^{1/2+\Theta(1)})$).

Towards generalizing the coin-tossing results, Gordon and Katz [37] 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. [8] 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

[3] 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., $O(\log^* n)$ [24].

[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.

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 contribution 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 [54,28]. Interestingly, a classic construction for (interactive) distributed key generation by Pedersen [51] in the honest majority setting was found by Gennaro et al. [31] to unintentionally permit adversarial bias, which the same authors later proved can be tolerated in a number of applications [32].

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 [13,39,47] and variants of Abe's verifiable mixnets [1,14], 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. Multi-party RSA modulus generation [20,21] 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. It was our initial intuition that $\Omega(n)$ rounds were

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

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 $O(\sqrt{n})$ 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 [10] and later one hundred participants [13]. 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 [37].

The Basic Idea. The transformation underlying our compiler uses the "player-simulation technique" that goes back to Bracha [15] and is widely used in the Byzantine agreement and MPC literature (e.g., [41,43]) as well as the "player-elimination framework" [34,33]. 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" [35] or the "size of domain or range" [8]. 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 [13]. 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 [38]. 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 [10,12], using "random beacons" as an external source of entropy [13], or considering restricted forms of *algebraic adversaries* [29,47] in the random oracle model.

Round-Robin Sampling Protocols. The powers-of-tau protocol has a simple structure shared by other (seemingly unrelated) protocols [1,14], 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 [13,39,47], this abstraction captures an additional interesting protocol from the literature: verifiable mixnets [14], 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., [13,47]) 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 [44]. 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 a (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* [42,25] 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* [6,53] 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* [46] and *restricted identifiable abort* [24].

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 adversary, 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. [44] 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) [11]. 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.

SNARKs with Updateable Setup. Several recent Succinct Non-interactive Arguments (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 [39,50,30,22]. Since their proofs already account for adversarial bias and the form of their trusted setup derives from the setup of Kate et al. [44], 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. [14], 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, the full version of this paper will include an additional, non-generic construction that specifically computes the powers-of-tau, and an analysis of its concrete costs. We give a summary of this additional result in Section 5.

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 $\mathbf{a} := [2,7]$ and $\mathbf{b} := \{1,3,4\}$, 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 $\mathsf{GenSID}(\mathsf{sid}, x, \mathtt{x})$ 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., [25]) and *cannot* be captured in the inherently asynchronous UC framework [17]. For concreteness, we will consider the synchronous UC modeling of Katz et al. [45], 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 [48] or Baum et al. [7].

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

1. If a record of the form $(\texttt{unbiased}, \mathsf{sid}, *)$ 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 broadcast $(\texttt{output}, \mathsf{sid}, y)$ 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

[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 [45].

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 $\mathcal{P}_1 \dots \mathcal{P}_n$ 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. 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 broadcast $(\texttt{output}, \mathsf{sid}, y)$ to all parties.

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).

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 broadcasted 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 [42]

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 $x_i \in \mathbb{X}_i$ 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 broadcast $(\texttt{abort}, \mathsf{sid}, c)$ to all parties. Otherwise, broadcast $(\texttt{output}, \mathsf{sid}, y)$ 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 3.12 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 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 3.11. After this, we give a folklore method for realizing $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]_{\mathsf{PV}}$ in Lemma 3.12, and use it to prove our main efficiency result in Corollary 3.13.

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 3.11. Since corruptions are static, a single simulator can be constructed that follows the code of either $\mathcal{S}_{\mathsf{Compiler}}$ or $\mathcal{S}_{\mathsf{CompilerPV}}$ 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 in the full version of this paper.

Proof Sketch. In this sketch 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 path and the functionality $[\![\mathcal{F}_{\mathsf{PostTrans}}(f, m, u, m)]\!]_{\mathsf{PV}}$.

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 $j := 1$, $k := 1$, $\mathbf{a}_0 := u$, $\kappa := \bot$.
2. Obtain a candidate $\mathbf{v}_k$ by sending either `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 $[m] \setminus \{\theta\}$ on behalf of $[\![\mathcal{F}_{\mathsf{SFE\text{-}IA}}]\!]^{\mathsf{PV}}$ to learn the values of $\hat{\mathbf{a}}_{k,i}$ for $i \in [m] \setminus \{\theta\}$.
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. □

The formal proof of Lemma 3.10 is given in the full version of this paper, where we also prove a similar lemma holds when there are no honest participants, but at least one honest verifier, and sketch a proof for the folklore construction of secure function evaluation with publicly verifiable identifiable abort.

Lemma 3.11. 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.

Lemma 3.12. (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 $\mathcal{F}_{\mathsf{OT}}$.

Corollary 3.13. 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 player-replaceable 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 3.12. □

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 make use of the standard UC NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$, originally formulated by Groth et al. [40]. 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 [27] 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 [52,40,18]. Regardless, we give our protocol description next.

Protocol 4.2. $\pi_{\mathsf{RRSample}}(f, n, u)$**. 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 $(\texttt{reject}, \mathsf{sid}, \mathsf{GenSID}(\mathsf{sid}, j))$ (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 $(\texttt{output}, \mathsf{sid}, \mathbf{v}_n)$ to the environment.[b]

Verification: If there is an observing verifier $\mathcal{V}$, then on receiving $(\texttt{observe}, \mathsf{sid})$ 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 $(\texttt{output}, \mathsf{sid}, \mathbf{v}_n)$ 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

In this section, we present the security theorem for the above protocol, a corollary concerning the application of our compiler under various generic realizations of $\mathcal{F}_{\mathsf{NIZK}}$, and a theorem stating that for the specific class of functions covered by Definition 4.1, the compiled protocol realizes the *original* functionality. Proofs of the theorems in this section are given in the full version of this paper.

Theorem 4.3. 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 $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$-hybrid model.

Corollary 4.4. 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 4.5.

Remark 4.5. $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}_f}$ is realizable for any polynomial-time f in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model under the existence of enhanced trapdoor permutations, or the existence of homomorphic trapdoor functions and the decisional linear assumption in a bilinear group, or the LWE assumption, or the LPN and DDH assumptions.

Theorem 4.6. 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.

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 4.7. 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. [40], or Canetti et al. [18]. Fortunately, the NIZK construction of De Santis et al. [52] requires only a uniform common random string. Thus we achieve our main theoretical result with respect to the powers of tau:

Corollary 4.8. 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 4.7 and Theorems 4.4 and 4.6 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 the Cleve bound [23]. 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 SNARKs [12,47], such proofs have thus far been monolithic and specific to the particular combination of SNARK and sampling scheme that they address. Moreover, because SNARKs are proven secure in powerful idealized models, prior distributed sampling protocols were analyzed in those models as well. Unlike SNARKs, which require knowledge assumptions, the security of the Kate et al. [44] polynomial-commitment scheme can be reduced to a concrete falsifiable assumption. This presents a clean, standalone context in which to examine the impact 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 [44, 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:

Definition 4.9. (d**-Strong Diffie-Hellman Assumption** [11])**.** Let the security parameter λ determine a group $\mathbb{G}$ of prime order q that is generated by G. For every PPT adversary $\mathcal{A}$,

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

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

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 $\mathcal{Z}$.

[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.

2. In the emulated experiment, on receiving $(\texttt{sample}, \mathsf{sid})$ 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 $(\texttt{output}, \mathsf{sid}, z = \{\tau \cdot G, \tau^2 \cdot G, \ldots, \tau^d \cdot G\})$ 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 4.11. **((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 $(\mathcal{Z}, \mathcal{A})$,

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

Note that per Canetti [17], 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. [44], 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 4.12. 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 $t' \approx t$.

The proof of the above theorem appears in the full version of this document.

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 SNARKs. The game-based notion of updateable security with respect to structured reference strings was defined recently by Groth et al. [39]. Informally, if a 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 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 $\mathcal{F}^{\mathcal{R}_f}_{\mathsf{NIZK}}$ for whatever

f maps the previous SRS to the next one, and the update algorithm of Groth et al. (also used by later works [50,30,22]) does not have straight-line extraction. Modifying any updateable SNARK to fit into our model is beyond the scope of this work. Nevertheless, we discuss two alternatives that do not involve modifying the SNARK.

First, we observe that if the proofs of well-formedness of the Groth et al. update procedure [39] 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 4.7. This implies a result similar to Corollary 4.8: 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 [22] and Plonk [30]:

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

whereas Sonic [50] 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 4.7.

Because 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. [14]. 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 [9] *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:

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

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 4.13. 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. □

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 4.14. For any prime-order group $\mathbb{G}$ and any $d \in \mathbb{N}^+$, $n, m \in \mathbb{N}^+$ such that $m \leq n$, $\mathsf{pk} \in \mathbb{G}$, and $\mathbf{c} \in \text{image}(\mathsf{Enc}_{\mathsf{pk}})^d$, 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 4.5.

Proof. By conjunction of Lemma 4.13 and Theorems 4.4 and 4.6. □

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 [14, Definition 7]. Furthermore, the game-based security definition of Boyle et al. [14, 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.

5 With Concrete Efficiency

The previous sections of this paper were 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, $d \in [2^{10}, 2^{20}]$ [13]. This implies that the $\mathsf{PowTau}_{\mathbb{G},d}$ function involves many thousands of elliptic curve scalar multiplications; if rendered into a boolean circuit, it 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. Here we will give a simple sketch and make a few high-level efficiency

claims. The full version of this paper contains a full protocol description, an in-depth concrete cost analysis, and a proof of security.

$\pi_{\mathsf{BilinearSRS}}$ 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 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.

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, and it natively supports arithmetic computations over an arbitrary field, which implies that this computation requires only $O(d)$ multiplication 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 EC 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 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 both the pairing operation and the scalar-multiplication operation have large concrete computational costs; thus we must use them judiciously. In $\pi_{\mathsf{BilinearSRS}}$, these two operations incur the vast majority of concrete computational costs that are not due to the protocol realizing $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$. We define a metric of the overall wall-clock latency incurred by EC pairings and, similarly, a metric of the latency incurred by EC scalar operations. For $\pi_{\mathsf{BilinearSRS}}$, the former cost is in $O(\sqrt{n})$ and the latter is in $O(d \cdot \sqrt{n} + n \cdot \lambda / \log \lambda)$ for active participants or $O(n + d \cdot \sqrt{n})$ for passive verifiers; this is an improvement upon the round-robin SRS sampling technique, which (after optimization) has a pairing latency in $O(n)$ and a scalar latency in $O(d \cdot n)$.

It should be noted that our protocol is not a strict improvement upon prior techniques in all respects: it requires $O(n^{1.5} \cdot d \cdot \lambda + n^{1.5} \cdot \lambda^2 / \log \lambda)$ bits to be broadcasted in total, *not including* the communication costs of the protocol that realizes $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$; in this respect our approach is strictly worse than prior work. The protocol that realizes $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}]\!]_{\mathsf{PV}}$ must evaluate $O(d \cdot n^{1.5})$ input and output gates and $O(d \cdot n)$ multiplication gates in total, among groups of $\sqrt{n}$ active participants. We identify this as our most significant concrete bottleneck. Substantial progress has recently been made toward optimizing generic MPC in the security-with-abort setting, but the publicly-verifiable identifiable-abort setting has received less attention thus far. We hope and expect that this will change, and $\pi_{\mathsf{BilinearSRS}}$ will move toward practicality as a result.

Acknowledgements

We thank Alon Rosen for a helpful discussion. We furthermore thank an anonymous reviewer for making us aware of certain practical optimizations used in the full version of this paper. 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

1. Masayuki Abe. Mix-networks on permutation networks. In *ASIACRYPT*, 1999.
2. Gilad Asharov. Towards characterizing complete fairness in secure two-party computation. In *TCC*, 2014.
3. Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. Complete characterization of fairness in secure two-party computation of Boolean functions. In *TCC*, 2015.
4. Gilad Asharov, Yehuda Lindell, and Tal Rabin. A full characterization of functions that imply fair coin tossing and ramifications to fairness. In *TCC*, 2013.
5. 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.
6. Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *SCN*, 2014.
7. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In *EUROCRYPT*, 2021.
8. Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. In *CRYPTO*, 2011.
9. Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT*, 2009.
10. Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *IEEE S&P*, 2015.

11. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, 2004.
12. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. In *FC*, 2018.
13. 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, 2017.
14. Elette Boyle, Saleet Klein, Alon Rosen, and Gil Segev. Securing Abe's mix-net against malicious verifiers via witness indistinguishability. In *SCN*, 2018.
15. Gabriel Bracha. An O(log n) expected rounds randomized Byzantine generals protocol. *JACM*, 34(4), 1987.
16. Niv Buchbinder, Iftach Haitner, Nissan Levi, and Eliad Tsfadia. Fair coin flipping: Tighter analysis and the many-party case. In *SODA*, 2017.
17. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
18. Ran Canetti, Pratik Sarkar, and Xiao Wang. Triply adaptive UC NIZK. *IACR Cryptol. ePrint Arch.*, 2020.
19. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777, 2019.
20. Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In *CRYPTO*, 2020.
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 *IEEE S&P*, pages 590–607, 2021.
22. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In *EUROCRYPT*, 2020.
23. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, 1986.
24. Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. From fairness to full security in multiparty computation. *Journal of Cryptology*, 35(1), 2022.
25. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4), 2017.
26. Dana Dachman-Soled. Revisiting fairness in MPC: polynomial number of parties and general adversarial structures. In *TCC*, 2020.
27. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, 2005.
28. Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. pages 300–316, 2001.
29. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO*, 2018.
30. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019.
31. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*, pages 295–310, 1999.
32. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of Pedersen's distributed key generation protocol. pages 373–390, 2003.

33. Oded Goldreich. *Foundations of Cryptography – VOLUME 2: Basic Applications*. Cambridge University Press, 2004.
34. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
35. S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. In *STOC*, 2008.
36. S. Dov Gordon and Jonathan Katz. Complete fairness in multi-party computation without an honest majority. In *TCC*, 2009.
37. S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. *Journal of Cryptology*, 25(1), 2012.
38. Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.
39. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In *CRYPTO*, 2018.
40. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *JACM*, 59(3), 2012.
41. Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1), 2000.
42. Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO*, 2014.
43. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, 2008.
44. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.
45. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, 2013.
46. 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, *EUROCRYPT*, 2016.
47. Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. *IACR Cryptol. ePrint Arch.*, 2021, 2021.
48. Chen-Da Liu-Zhang and Ueli Maurer. Synchronous constructive cryptography. In *TCC*, 2020.
49. Nikolaos Makriyannis. On the classification of finite Boolean functions up to fairness. In *SCN*, 2014.
50. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *CCS*, 2019.
51. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *EUROCRYPT*, 1991.
52. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, 2001.
53. Berry Schoenmakers and Meilof Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In *ACNS*, 2015.
54. Markus Stadler. Publicly verifiable secret sharing. In *EUROCRYPT*, pages 190–199, 1996.