Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs

Anonymized for submission to Security and Privacy 2015

***Abstract*—Non-interactive zero-knowledge proofs (NIZKs) are a powerful cryptographic tool, with numerous potential applications. However, succinct NIZKs (e.g., zk-SNARK schemes) necessitate a trusted party to generate and publish some public parameters, to be used by all provers and verifiers. This party is trusted to *correctly* run a probabilistic algorithm (specified by the the proof system) that outputs the public parameters, and publish them, without *leaking* any other information (such as the internal randomness used by the algorithm); violating either requirement may allow producing convincing "proofs" of false statements. This trust requirement poses a serious impediment to deploying NIZKs in many applications, because a party that is trusted by all users of the envisioned system may simply not exist.**

In this work, we show how public parameters for a class of NIZKs can be generated by a multi-party protocol, such that if at least one of the parties is honest, then the result is secure (in both aforementioned senses) and can be subsequently used for generating and verifying numerous proofs without any further trust. We design and implement such a protocol, tailored to efficiently support the state-of-the-art NIZK constructions with short and easy-to-verify proofs (Parno et al. IEEE S&P '13; Ben-Sasson et al. USENIX Sec '14; Danezis et al. ASIACRYPT '14). Applications of our system include generating public parameters for systems such as Zerocash (Ben-Sasson et al. IEEE S&P '13) and the scalable zero-knowledge proof system of (Ben-Sasson et al. CRYPTO '14).

I. Introduction

Consider first the following simple scenario. A server owns a private database x, and a client wishes to learn $y := F(x)$ for a public function F, selected either by himself or someone else. A (hiding) commitment cm to x is known publicly. For example, x may be a database containing genetic data, and F may be a machine-learning algorithm that uses the genetic data to compute a classifier y. On the one hand, the client seeks *integrity of computation*: he wants to ensure that the server reports the correct output y (e.g., because the classifier y will be used for critical medical decisions). On the other hand, the server seeks *confidentiality* of his own input: he is willing to disclose y to the client, but no additional information about x beyond y (e.g., because the genetic data x may contain sensitive personal information).

Zero-knowledge proofs. Achieving the combination of the above security requirements seems paradoxical: the client does not have the input x, and the server is not willing to share it. Yet, cryptography offers a powerful tool that is able to do just that: *zero-knowledge proofs* [1]. The server, acting as the prover, attempts to convince the client, acting as the verifier, that the following NP statement is true: "there is $\tilde{x}$ such that $y = F(\tilde{x})$ and $\tilde{x}$ is a decommitment of cm". Indeed: (a) the proof system's *soundness* property addresses the client's integrity concern, because it guarantees that, if the NP statement is false, the prover cannot convince the verifier (with high probability);[1] and (b) the proof system's *zero-knowledge* property addresses the server's confidentiality concern, because it guarantees that, if the NP statement is true, the prover can convince the verifier without leaking any information about x (beyond was is leaked by y).

Non-interactivity. While zero-knowledge proofs can address the above simple scenario, they also apply more widely, including to scenarios that involve many parties who do not trust each other or are not all simultaneously online. In such cases, it is desirable to use *non-interactive zero-knowledge proofs* (NIZKs), where the proof consists of a single message π that can be verified by anyone. For example, such a proof π can be stored for later use, or it can be verified by multiple parties without requiring the prover to separately interact with each of these.

Unfortunately, NIZKs do not exist for languages outside BPP (even when soundness is relaxed to hold only computationally) [3], [4]. But, if a trusted party is available for a one-time setup phase, then, under suitable hardness assumptions, NIZKs exist for all languages in NP [5], [6], [7], [8]. During the setup phase, the trusted party runs a probabilistic polynomial-time *generator* algorithm G (prescribed by the proof system) and publishes its output pp, called the *public parameters*; afterwards, the trusted party is no longer needed, and anyone can use pp to produce proofs or to verify them. Soundness of the NIZK depends on this trusted setup: if pp is not correctly generated, or if secret internal

[1]Sometimes a property stronger than soundness is required: *proof of knowledge* [1], [2], which guarantees that, whenever the verifier is convinced, not only can he deduce that a witness exists, but also that the prover *knows* one such witness.

randomness used within G is revealed, then it may be feasible to convince that verifier that false NP statements are true.

The problem of parameter generation. If no trusted party is available, how are the public parameters pp generated? One approach is to look for, in Nature or Society, a publicly-observable distribution that equals (or is close to) pp's distribution. For example, if G merely outputs a random binary string of a certain length,[2] it may be possible, via suitable measurements and post-processing of, e.g., data about sun spots or the stock market, to extract bits that are close to random. (See [9], [10] for work in this direction, and [11] for a NIST prototype using quantum randomness sources). However, if G follows a more complex probabilistic strategy, then there may be no stochastic process in Nature or Society that yields a distribution close to pp's.

An attractive alternative approach to address the problem of parameter generation is the following:

construct a multi-party protocol for securely generating the public parameters pp.

The setup phase will then involve a large set of parties running the multi-party protocol for generating pp, and for soundness of the NIZK to hold it will suffice that only a few (ideally, even just one) of these parties are honest. Clearly, this is a weaker and more realistic trust assumption then placing ultimate trust in any single party.

Several works have explored this approach for the parameter distributions of various cryptographic primitives and, more generally, one can invoke secure multi-party computation [12], [13] to obtain a feasibility result. Yet, as discussed in Section II, prior works do not yield satisfactory efficiency in our setting, which we now introduce.

A. Our focus

The focus of our work is to address the parameter generation problem for a particular kind of NIZKs: *zero-knowledge succinct arguments of knowledge* (zk-SNARKs) [14], [15], [16]; these are NIZKs for which the proof is short and easy to verify [17]. Concretely, our goal is to obtain efficient multi-party protocols for securely sampling the public parameters required by zk-SNARKs, as we now explain.

zk-SNARK constructions. There are many zk-SNARK constructions, with different properties in efficiency and supported languages. In *preprocessing zk-SNARKs*, the complexity of sampling public parameters grows with the size of the computation being proved [18], [19], [16], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31]; in *fully-succinct zk-SNARKs*, that complexity is independent of computation size [17], [32], [33], [34], [15], [35], [36], [37], [38], [39], [40]. Working prototypes have been achieved for preprocessing zk-SNARKs [21], [22], [25], [27], [30] and fully-succinct ones [39]. Several works have also explored various applications of zk-SNARKs [41], [42], [43], [44], [45].

Public parameters of zk-SNARKs. Despite the aforementioned multitude of constructions, Bitansky et al. [16] showed that essentially all known preprocessing zk-SNARK constructions can be "explained" as the combination of a *linear interactive proof* (LIP) and an encoding that forbids all but linear homomorphisms. This yields a unified view of parameter generation across preprocessing zk-SNARKs (that are not fully succinct). Namely, given an NP relation $\mathcal{R}$, the generator G adheres to the following computation pattern when producing public parameters for $\mathcal{R}$: (i) derive from $\mathcal{R}$ a certain circuit C;[3] (ii) evaluate C at a random input; (iii) output the encoding of the evaluation. In other words, public parameters of preprocessing zk-SNARKs are the encodings of random evaluations of certain circuits.

The sampling problem. Concretely, for a prime r, the circuit C is defined over a size-r field $\mathbb{F}_r$ and the encoding of $\alpha \in \mathbb{F}_r$ is $\alpha \cdot \mathcal{G}$, where $\mathcal{G}$ generates an order-r group $\mathbb{G}$. Moreover, $\mathbb{G}$ is a *duplex-pairing group* (i.e., $\mathbb{G}$ is a subgroup of some $\mathbb{G}_1 \times \mathbb{G}_2$ equipped with a pairing). This discussion motivates the following multi-party sampling problem:

> Let r be a prime, $\mathbb{G} = \langle \mathcal{G} \rangle$ an order-r group, n a positive integer, and $C \colon \mathbb{F}_r^m \to \mathbb{F}_r^h$ an $\mathbb{F}_r$-arithmetic circuit. Construct an n-party protocol for securely sampling $\mathsf{pp} := C(\vec{\alpha}) \cdot \mathcal{G}$ for random $\vec{\alpha}$ in $\mathbb{F}^m$.

We thus seek a multi-party protocol such that, even when all but one of the parties are malicious, the protocol's output is pp sampled from the correct distribution and, moreover, parties learn nothing beyond pp itself. We study this problem, and the special case of generating public parameters for preprocessing zk-SNARKs.

B. Our contributions

We design, build, and evaluate a multi-party protocol for securely sampling encodings of random evaluations of certain circuits. The resulting system enables us, in particular, to sample the public parameters for a class of

[2]NIZKs for which G outputs a random binary string are said to be in the *common random string* model.

[3]The circuit C computes the LIP's verifier's message. In some LIP constructions, C can be interpreted as an arithmetization of the decision algorithm of the relation $\mathcal{R}$.

preprocessing zk-SNARKs that includes [21], [25], [31]; we integrated our system with `libsnark` [46], a C++ zk-SNARK library, to facilitate this application. In more detail, we present the following two main contributions.

(1) Secure sampling for a class of circuits. We design, build, and evaluate a multi-party protocol that securely samples values of the form $C(\vec{\alpha}) \cdot \mathcal{G}$ for a random $\vec{\alpha}$, provided that C belongs to a certain circuit class $\mathbf{C}^\star$. Roughly, $\mathbf{C}^\star$ comprises the arithmetic circuits such that, for every gate, (i) the gate's output is also a circuit's output, and (ii) at least one of the gate's two inputs is also a circuit's input.

The multi-party protocol runs atop a synchronous network with an authenticated broadcast channel and a common random string. The computation proceeds in rounds and, at each round, the protocol's schedule determines which parties act; a party acts by broadcasting a message to all other parties.[4]

When n parties participate, our protocol is secure against $\leq n-1$ malicious parties. If even one of the parties is honest, and assuming the protocol reaches completion, then the protocol's output is a sample from the designated distribution and no other information leaks.[5] Each party runs in time $O_\lambda(\mathsf{size}(C))$, where $O_\lambda(\cdot)$ hides a fixed polynomial in the security parameter λ. The number of rounds is $n \cdot \mathsf{depth}^\star(C) + O(1)$ and the number of broadcast messages is $O(n \cdot \mathsf{depth}^\star(C))$. Here, $\mathsf{depth}^\star(C)$ denotes the $\star$-*depth* of C, which is at most the (standard) circuit depth of C, but sometimes much smaller, as is (crucially) the case for the zk-SNARK application discussed below.

While the above results hold for any group $\mathbb{G}$, our implementation is specialized to the case when $\mathbb{G}$ is duplex-pairing because, for this case, several additional optimizations are possible.

(2) Application to zk-SNARKs. Our system can be used to securely sample public parameters of a zk-SNARK, whenever zk-SNARK's generator can be cast as sampling the encoding of the random evaluation of a circuit that lies in the class $\mathbf{C}^\star$. While the class $\mathbf{C}^\star$ appears restrictive, we observe that several known constructions of preprocessing zk-SNARK have such a generator.

To facilitate this application to zk-SNARKs, we (i) integrated our system with `libsnark` [46], and (ii) applied our system to generating public parameters for two specific zk-SNARK constructions: that of [21], [25] and that of [31]. The first construction supports proving the satisfiability of arithmetic circuits, while the second one supports proving that of boolean circuits.[6] (We also extended `libsnark` with an implementation of [31]'s zk-SNARK because, previously, `libsnark` provided only a zk-SNARK based on [21], [25].)

Given an arithmetic circuit D, our code generates a related circuit C_{PGHR} in $\mathbf{C}^\star$, such that the encoding of a random evaluation of C_{PGHR} corresponds to public parameters for [21], [25]'s zk-SNARK when proving satisfiability of D. If D has N_{w} wires and N_{g} gates, then C_{PGHR} has size $11 \cdot N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 38$ and $\star$-depth 3. Similarly, given a boolean circuit D, our code generates a related circuit C_{DFGK} in $\mathbf{C}^\star$ for [31]'s zk-SNARK; if D has N_{w} wires and N_{g} gates, then C_{DFGK} has size $2 \cdot N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 10$ and $\star$-depth 2.

We evaluate the concrete costs of our protocol when used to generate the public parameters, needed by aforementioned zk-SNARK, in order to prove satisfiability of specific circuits D that arise in specific applications:

- Our system can securely generate the public parameters for Zerocash [44], a decentralized anonymous payment systems extending Bitcoin. Letting D be the circuit that implements the NP relation used in Zerocash: C_{PGHR} has size 138467206 and $\star$-depth 3; in our multi-party protocol, the number of rounds is $3 \cdot n + 3$ and each party works for 14124 s.
- Our system can securely generate the public parameters needed for the scalable zk-SNARK of [39], which proves correct execution of programs on a 32-bit RISC architecture. Letting D be the circuit used in [39]: C_{PGHR} has size 8027609 and $\star$-depth 6; in our multi-party protocol, the number of rounds is $6 \cdot n + 6$ and each party works for 4048 s. (More precisely, [39] requires *two* circuits, and here and elsewhere we present the sum of the costs for each complexity measure.)

C. Summary of challenges and techniques

We describe at high level the challenges that arise, as well as the techniques that we employed to address them, for each of our two main contributions.

1) Secure sampling for a class of circuits

Let r be a prime, $\mathbb{G} = \langle \mathcal{G} \rangle$ an order-r group, n a positive integer, and $C \colon \mathbb{F}_r^m \to \mathbb{F}_r^h$ an $\mathbb{F}_r$-arithmetic circuit. We

[4]A broadcast channel can also be thought of as an append-only public logbook, such as the one that Bitcoin seeks to realize via its puzzle-based block-chain protocol [47]. Authentication can be achieved, e.g., via digital signatures supported by a public-key infrastructure.

[5]A malicious party may prevent the protocol from reaching completion, by acting incorrectly or by delaying their prescribed broadcasts. However, the culprit party can be readily identified.

[6]More precisely, both actually support more general NP-complete relations, but this technical detail is not important for the present discussion.

seek a multi-party protocol for securely sampling $C(\vec{\alpha}) \cdot \mathcal{G}$ for a random $\vec{\alpha}$. We are willing to compromise on functionality by restricting C to belong to a yet-to-be-determined circuit class $\mathbf{C}^{\star}$, provided that, in turn, we gain improved concrete efficiency (since, ultimately, we want to implement the protocol and use it to generate zk-SNARK public parameters).

The ideal functionality. A natural solution approach is to construct a multi-party protocol that securely implements the ideal functionality $f_{C,\mathcal{G}}$ defined as follows. On input $\vec{\sigma} := (\vec{\sigma}_1, \ldots, \vec{\sigma}_n)$ where $\vec{\sigma}_i = (\sigma_{i,1}, \ldots, \sigma_{i,m}) \in \mathbb{F}_r^m$ is party i's input, $f_{C,\mathcal{G}}$ first computes $\alpha_j := \prod_{i=1}^{n} \sigma_{i,j}$ for $j = 1, \ldots, m$; then $f_{C,\mathcal{G}}$ sets $\vec{\alpha} := (\alpha_1, \ldots, \alpha_m)$ and computes $\vec{\mathcal{P}} := C(\vec{\alpha}) \cdot \mathcal{G}$; finally, $f_{C,\mathcal{G}}$ outputs $\vec{\mathcal{P}}$. Indeed, if at least one party honestly provides an input consisting of random elements, $f_{C,\mathcal{G}}$ outputs the encoding of a random evaluation of C.[7]

However, we now argue that one is unlikely to obtain good concrete efficiency by using off-the-shelf solutions to securely implement the ideal functionality $f_{C,\mathcal{G}}$. Briefly, this is because the "core" of $f_{C,\mathcal{G}}$ is the computation of $C(\vec{\alpha}) \cdot \mathcal{G}$, which involves both the evaluation of an $\mathbb{F}_r$-arithmetic circuit and scalar multiplications over the group $\mathbb{G}$.

Some approaches to implement $f_{C,\mathcal{G}}$. On the one hand, we could express the aforementioned computation via a boolean circuit, and invoke a suitable secure computation protocol for boolean circuits (see Section II). However, the conversion to a boolean circuit is expensive. For example, the number of boolean gates required to compute $C(\vec{\alpha})$ alone is a factor of $\log_2 r$ larger than the number of $\mathbb{F}_r$-arithmetic gates for the same task, because each addition and multiplication in $\mathbb{F}_r$ is expanded into a boolean sub-circuit of size $\geq \log_2 r$. We expect that $\log_2 r \geq 100$ (since r is a cryptographically-large prime), which results in a blowup of two orders of magnitude in the number of gates.

On the other hand, we could express the computation of $C(\vec{\alpha}) \cdot \mathcal{G}$ via an arithmetic circuit, and invoke a suitable secure computation protocol for arithmetic circuits (see Section II). However, over what field should the arithmetic circuit be defined? While C is defined over the field $\mathbb{F}_r$, the group $\mathcal{G}$ may not be. In fact, for the application considered in this paper (namely, secure sampling of public parameters for zk-SNARKs), the group $\mathbb{G}$ is defined over a prime field $\mathbb{F}_q$ that is *different* from $\mathbb{F}_r$. If we express the computation as an $\mathbb{F}_r$-arithmetic circuit then, while evaluating C may be efficient, scalar multiplications over $\mathbb{G}$ may not be. Conversely, if we express the computation as an $\mathbb{F}_q$-arithmetic circuit, while scalar multiplications over $\mathbb{G}$ may be efficient, evaluating C may not be. In either case, we find ourselves with the overheads associated to "characteristic simulation", which also are on the order of $\log_2 r$ or $\log_2 q$.

One approach to address the above issue could be to extend existing protocols to support multiple characteristics. For example, one idea is to first compute $\vec{\beta} := C(\vec{\alpha})$ using a protocol that supports $\mathbb{F}_r$-arithmetic circuits and then, somehow, instead of having parties decommit and obtain $\vec{\beta}$, use the commitment to $\vec{\beta}$ as an input to a second protocol, which instead supports $\mathbb{F}_q$-arithmetic circuits, to compute $\vec{\beta} \cdot \mathcal{G}$ by interpreting $\vec{\beta}$ in binary and conducting scalar multiplications in $\mathbb{G}$ as needed. We did not investigate whether such an approach leads to a feasible protocol and, instead, opted for an alternative, and more direct, approach, which we outline next.

Our approach to implement $f_{C,\mathcal{G}}$. We observe that, while the envisioned application to zk-SNARKs does not restrict C in any way, for particular zk-SNARK constructions (including [21], [25], [31]) the circuit C can be written to be of a special form (with some effort, see Section I-C2 below). Specifically, for every gate in C, the gate's output is also an output of the circuit and, moreover, the gate is either a linear combination or multiplication for which one of the two wires is also an input wire. We call $\mathbf{C}^{\star}$ the class of such circuits, and restrict our attention to implementing $f_{C,\mathcal{G}}$ provided that $C \in \mathbf{C}^{\star}$.

Next, for such circuits, we design a protocol that enables parties to jointly homomorphically evaluate the circuit C (avoiding, in particular, first computing $\vec{\beta} := C(\vec{\alpha})$ and then $\vec{\beta} \cdot \mathcal{G}$). Roughly, first all parties commit to their shares; then, for each multiplication gate, since one of the two gate's inputs is also an input to the circuit, every party can, in sequence, contribute, and prove correct contribution of, his share of the input. (As for linear combinations, they are "for free" as in many other multi-party protocols.)

A naive realization of the above strategy yields an enormous number of rounds: n times C's depth. In contrast, we show that, via a careful choice of when each party contributes his own sure, we can reduce the number of rounds to only n times C's $\star$-depth, where $\star$-depth is a much milder notion of depth.

We realize the above approach by splitting the construction in two steps. First, a reduction from the problem of sampling the encoding of a random evaluation of C

[7]More precisely, $f_{C,\mathcal{G}}$ must also check that none of the parties' inputs contains a zero. Doing so biases the output distribution, but only negligibly so because r has cryptographic size. Thus, this technical detail does not cause any problems.

to the problem of jointly evaluating a related circuit $\tilde{C}$. Second, a protocol for evaluating $\tilde{C}$. The steps greatly simplify providing a formal proof of security.

Our implementation is specialized to the case when $\mathbb{G}$ is a duplex-pairing group, in which case the NIZKs used by parties can be implemented very efficiently via Schnorr proofs.

2) *Application to zk-SNARKs*

We wish to apply our system to generating public parameters for two specific zk-SNARK constructions: that of [21], [25] and that of [31]. This requires writing code that transforms the generator's input NP relation, represented as an instance D of arithmetic or boolean satisfiability, into a corresponding circuit C in $\mathbf{C}^\star$ such that $C(\vec{\alpha}) \cdot \mathcal{G}$ for a random $\vec{\alpha}$ equals the suitable distribution of public parameters for each of the two constructions.

Constructing an efficient transformation from D to C, for either of the constructions, is not straightforward, because of the requirement that C is in $\mathbf{C}^\star$ (which allows us to invoke our secure sampling multi-party protocol). Briefly, in both cases, an issue that arises is how to construct a sub-circuit that given an input τ, evaluates all Lagrange polynomials (defined over a certain subset of $\mathbb{F}_r$; indeed, the standard linear-size circuit for this operation involves division gates, which our protocol does not handle (and are thus not included in $\mathbf{C}^\star$). Instead of relying on the standard circuit, we rely on a suitable FFT-like sub-circuit that avoids the division gates.

II. Prior work

The problem of setting up public parameters for NIZKs has been studied before, especially in the setting where the public parameters merely consist of a random string. For example, [48], [9], [10] study various aspects of this problem. There are also other cryptographic primitives that require a set of public parameters to be known to every party in the system, and various works have explored distributed generation of such parameters for various distributions [49], [50], [51], [52], [53], [54].

Secure multi-party computation protocol that work against malicious majorities directly give feasibility results. As discussed in Section I-C, these feasibility results seem unlikely to give good concrete efficiency in practice for the particular ideal functionality that we are interested in. This includes recent state-of-the-art protocols that support arithmetic circuits [55], [56], because in our case we need simultaneous support for arithmetic circuits defined over two different fields.

III. Definitions

We give the definitions needed for technical discussions. Throughout, we denote by λ the security parameter. The input 1^λ is implicit to all cryptographic algorithms that we consider. We let $f = O_\lambda(g)$ mean there exists $c > 0$ such that $f = O(\lambda^c g)$.

A. *Basic notation*

Vectors are denoted by arrow-equipped letters (such as $\vec{a}$); their entries carry an index but not the arrow (e.g., a_1 or a_2). Concatenation of vectors (and scalars) is denoted by the operator $\circ$. We write $\{y \,|\, x_1 \leftarrow D_1\,;\, x_2 \leftarrow D_2\,;\, \ldots\}_E$ to denote the distribution over y obtained by conditioning on the event E and sampling x_1 from D_1, x_2 from D_2, and so on, and then computing $y := y(x_1, y_2, \ldots)$. Given two distributions D and D', we write $D \overset{\mathsf{negl}}{=} D'$ to denote that the statistical distance between D and D' is negligible in a security parameter λ. A distribution D is efficiently sampleable if there exists a probabilistic polynomial-time algorithm A whose output follows the distribution D.

Groups. We denote by $\mathbb{G}$ a group, and only consider cyclic groups having a prime order r. Group elements are denoted with calligraphic letters (e.g. $\mathcal{P}, \mathcal{Q}$). We write $\mathbb{G} = \langle \mathcal{G} \rangle$ to denote that the element $\mathcal{G}$ generates $\mathbb{G}$, and use additive notation for group arithmetic. That is, $\mathcal{P} + \mathcal{Q}$ denotes addition of the two elements $\mathcal{P}$ and $\mathcal{Q}$; $a \cdot \mathcal{P}$ denotes scalar multiplication of $\mathcal{P}$ by the integer scalar a; and $\mathcal{O} := 0 \cdot \mathcal{P}$ denotes the identify element. (Since $r \cdot \mathcal{P} = \mathcal{O}$, we can equivalently think of the scalar a as belonging to the field of size r.) Given a vector $\vec{a} = (a_1, \ldots, a_n)$, we use $\vec{a} \cdot \mathcal{P}$ as a shorthand for the vector $(a_1 \cdot \mathcal{P}, \ldots, a_n \cdot \mathcal{P})$.

Fields. We denote by $\mathbb{F}$ a field, and by $\mathbb{F}_n$ the field of size n. We assume familiarity with prime-order fields; for background, see the book of Lidl and Niederreiter [57].

B. *Hiding commitments*

A *hiding commitment scheme* is a pair $\mathsf{COMM} = (\mathsf{COMM.Gen}, \mathsf{COMM.Ver})$ with the following syntax.

- $\mathsf{COMM.Gen}(x) \to (\mathsf{cm}, \mathsf{trap})$*:* On input data x, the *commitment generator* $\mathsf{COMM.Gen}$ probabilistically samples a commitment cm of x and a corresponding trapdoor trap.
- $\mathsf{COMM.Ver}(x, \mathsf{cm}, \mathsf{trap}) \to b$*:* On input data x, commitment cm, and trapdoor trap, the *commitment verifier* $\mathsf{COMM.Ver}$ outputs $b = 1$ if cm is a valid commitment of x with respect to the trapdoor trap (and $b = 0$ otherwise).

We will use a commitment scheme COMM that satisfies the standard completeness, (computational) binding, and (statistical) hiding properties.

C. Non-interactive zero-knowledge proofs of knowledge

A *non-interactive zero-knowledge proof of knowledge* (NIZK) for an NP relation $\mathscr{R}$ in the common random string model is a tuple $\mathsf{NIZK}_{\mathscr{R}} = (\mathsf{NIZK}_{\mathscr{R}}.\mathsf{P}, \mathsf{NIZK}_{\mathscr{R}}.\mathsf{V}, \mathsf{NIZK}_{\mathscr{R}}.\mathsf{E}, \mathsf{NIZK}_{\mathscr{R}}.\mathsf{S})$ with the following syntax.

- $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{P}(\mathsf{crs}, \mathbb{x}, \mathbb{w}) \to \pi$*:* On input common random string crs, instance $\mathbb{x}$, and witness $\mathbb{w}$, the *prover* $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{P}$ outputs a non-interactive proof π for the statement "there is $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathscr{R}$".
- $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{V}(\mathsf{crs}, \mathbb{x}, \pi) \to b$*:* On input common random string crs, instance $\mathbb{x}$, and proof π, the *verifier* $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{V}$ outputs $b = 1$ if π is a convincing proof for the statement "there is $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathscr{R}$".

Above, crs is a random string of $O_\lambda(1)$-bit (the exact length is prescribed by $\mathsf{NIZK}_{\mathscr{R}}$). The remaining two components are each pairs of algorithms, as follows.

- $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{E}_1 \to (\mathsf{crs}_{\mathrm{ext}}, \mathsf{trap}_{\mathrm{ext}})$*:* The *extractor's generator* $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{E}_1$ samples a string $\mathsf{crs}_{\mathrm{ext}}$ (indistinguishable from crs) and a corresponding trapdoor $\mathsf{trap}_{\mathrm{ext}}$. $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{E}_2(\mathsf{crs}_{\mathrm{ext}}, \mathsf{trap}_{\mathrm{ext}}, \mathbb{x}, \pi) \to \mathbb{w}$*:* On input $\mathsf{crs}_{\mathrm{ext}}$, $\mathsf{trap}_{\mathrm{ext}}$, instance $\mathbb{x}$, and proof π, the *extractor* $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{E}_2$ outputs a witness $\mathbb{w}$ for the instance $\mathbb{x}$.
- $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{S}_1 \to (\mathsf{crs}_{\mathrm{sim}}, \mathsf{trap}_{\mathrm{sim}})$*:* The *simulator's generator* $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{S}_1$ samples a string $\mathsf{crs}_{\mathrm{sim}}$ (indistinguishable from crs) and a corresponding trapdoor $\mathsf{trap}_{\mathrm{sim}}$. $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{S}_2(\mathsf{crs}_{\mathrm{sim}}, \mathsf{trap}_{\mathrm{sim}}, \mathbb{x}) \to \pi$*:* On input $\mathsf{crs}_{\mathrm{sim}}$, $\mathsf{trap}_{\mathrm{sim}}$, and instance $\mathbb{x}$ (for which there is $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathscr{R}$), the *simulator* $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{S}_2$ outputs π that is indistinguishable from an "honest" proof.

We will use a scheme $\mathsf{NIZK}_{\mathscr{R}}$ that satisfies the standard completeness, (computational and adaptive) proof-of-knowledge, and (statistical, adaptive, and multi-theorem) zero-knowledge properties. Note that the NIZKs we use here are for statements that are short and of a special form. Thus we can use specialized NIZK constructions that require merely a common random string, rather than a (structured) common reference string whose generation is the problem we set out to solve in the first place.

D. Arithmetic circuits

We consider *arithmetic*, rather than boolean, circuits. Given a field $\mathbb{F}$, an $\mathbb{F}$*-arithmetic circuit* C takes as input elements in $\mathbb{F}$, and its gates output elements in $\mathbb{F}$. We write $C \colon \mathbb{F}^m \to \mathbb{F}^h$ if C takes m inputs and produces h outputs.

Wires, inputs, gates, and size. We denote by $\mathsf{wires}(C)$ and $\mathsf{gates}(C)$ the wires and gates of C; also, we denote by $\mathsf{inputs}(C)$ and $\mathsf{outputs}(C)$ the subsets of $\mathsf{wires}(C)$ consisting of C's input and output wires. We denote by $\#\mathsf{wires}(C)$, $\#\mathsf{gates}(C)$,$\#\mathsf{inputs}(C)$, and $\#\mathsf{outputs}(C)$ the cardinalities of $\mathsf{wires}(C)$, $\mathsf{gates}(C)$, $\mathsf{inputs}(C)$, and $\mathsf{outputs}(C)$ respectively. The size of C is $\mathsf{size}(C) := \#\mathsf{inputs}(C) + \#\mathsf{gates}(C)$.

Bilinear gates. Each gate g of C is *bilinear*, i.e., g computes $(\alpha_0^{\mathsf{L}} + \sum_{j=1}^{d^{\mathsf{L}}} \alpha_j^{\mathsf{L}} \mathsf{w}_j^{\mathsf{L}}) \cdot (\alpha_0^{\mathsf{R}} + \sum_{j=1}^{d^{\mathsf{R}}} \alpha_j^{\mathsf{R}} \mathsf{w}_j^{\mathsf{R}}) \to \mathsf{w}$, where $\mathsf{L\text{-}coeffs}(g) := (\alpha_j^{\mathsf{L}})_{j=0}^{d^{\mathsf{L}}}$ are the left coefficients, $\mathsf{R\text{-}coeffs}(g) := (\alpha_j^{\mathsf{R}})_{j=0}^{d^{\mathsf{R}}}$ the right coefficients, $\mathsf{L\text{-}inputs}(g) := \{\mathsf{w}_j^{\mathsf{L}}\}_{j=1}^{d^{\mathsf{L}}}$ the left input wires, $\mathsf{R\text{-}inputs}(g) := \{\mathsf{w}_j^{\mathsf{R}}\}_{j=1}^{d^{\mathsf{R}}}$ the right input wires, and $\mathsf{output}(g) := \mathsf{w}$ the output wire. Bilinear gates include addition, multiplication, and constant gates. (As usual, the dependency graph induced by C's gates is acyclic.)

Further notions for circuits with a split domain. We also consider $\mathbb{F}$-arithmetic circuits C for which the inputs are partitioned into n disjoint sets; in such a case, we write $C \colon \mathbb{F}^{m_1} \times \cdots \times \mathbb{F}^{m_n} \to \mathbb{F}^h$ to express that the first m_1 inputs are in the first set, the next m_2 in the second, and so on; the integers $m_1, \ldots, m_n$ are then also part of C's description.

For $i = 1, \ldots, n$: we denote by $\mathsf{inputs}(C, i)$ the input wires that belong to the i-th set, and by $\mathsf{gates}(C, i)$ the gates that take as input an input wire in $\mathsf{inputs}(C, i)$; the notations $\#\mathsf{inputs}(C, i)$ and $\#\mathsf{gates}(C, i)$ denote the cardinalities of these sets; and we define $\mathsf{size}(C, i) := \#\mathsf{inputs}(C, i) + \#\mathsf{gates}(C, i)$.

Moreover, for any gate g in $\mathsf{gates}(C)$, $\mathsf{L\text{-}deps}(g)$ is the set of integers i in $\{1, \ldots, n\}$ for which there is a wire in $\mathsf{L\text{-}inputs}(g)$ whose value depends (topologically) on the value of a wire in $\mathsf{inputs}(C, i)$. Similarly, $\mathsf{R\text{-}deps}(g)$ is defined relative to $\mathsf{R\text{-}inputs}(g)$. We also define $\mathsf{deps}(g) := \mathsf{L\text{-}deps}(g) \cup \mathsf{R\text{-}deps}(g)$.

Two classes of circuits. We consider the following two classes of circuits.

- We denote by $\mathbf{C}^\star$ the class of $\mathbb{F}$-arithmetic circuits $C \colon \mathbb{F}^m \to \mathbb{F}^h$ for which every gate g in $\mathsf{gates}(C)$ is such that: (i) $\mathsf{output}(g) \in \mathsf{outputs}(C)$; (ii) $\mathsf{L\text{-}inputs}(g) \cap \mathsf{inputs}(C) = \emptyset$; (iii) $\mathsf{R\text{-}inputs}(g)$ is empty or a singleton in $\mathsf{inputs}(C)$.
- We denote by $\mathbf{C}^\dagger$ the class of $\mathbb{F}$-arithmetic circuits $C \colon \mathbb{F}^{m_1} \times \cdots \times \mathbb{F}^{m_n} \to \mathbb{F}^h$ for which every gate g in $\mathsf{gates}(C)$ is such that: (i) $\mathsf{output}(g) \in \mathsf{outputs}(C)$; (ii) $\mathsf{L\text{-}inputs}(g) \cap \mathsf{inputs}(C) = \emptyset$; (iii) $|\mathsf{R\text{-}deps}(g)| \leq 1$.

Notions of depth. For circuits in the classes $\mathbf{C}^\star$ and $\mathbf{C}^\dagger$, we work with alternative notions of depth, denoted $\star$*-depth* and $\dagger$*-depth*, that are defined as follows.

- Given C in $\mathbf{C}^\star$, $\mathsf{depth}^\star(C) := \max_{\mathsf{w}\in\mathsf{outputs}(C)} \mathsf{depth}^\star(\mathsf{w})$ and
$$\mathsf{depth}^\star(\mathsf{w}) := \begin{cases} 1 & \text{if } \mathsf{w} \in \mathsf{inputs}(C) \\ b_{\mathsf{w}} + \max \; \{\mathsf{depth}^\star(\mathsf{w}')\}_{\mathsf{w}'\in\mathsf{L\text{-}inputs}(g_{\mathsf{w}})} & \text{if } \mathsf{w} \notin \mathsf{inputs}(C) \end{cases}$$
where
  - g_{w} is the gate in $\mathsf{gates}(C)$ for which $\mathsf{w} = \mathsf{output}(g_{\mathsf{w}})$,
  - $b_{\mathsf{w}} \in \{0,1\}$ equals 1 if and only if either $|\mathsf{L\text{-}inputs}(g_{\mathsf{w}})| \geq 2$ or $|\mathsf{L\text{-}inputs}(g_{\mathsf{w}})| = 1 \wedge \mathsf{L\text{-}coeffs}(g)[0] \neq 0$.
- Given C in $\mathbf{C}^\dagger$, $\mathsf{depth}^\dagger(C) := \max_{\mathsf{w}\in\mathsf{outputs}(C)} \mathsf{depth}^\dagger(\mathsf{w})$ and
$$\mathsf{depth}^\dagger(\mathsf{w}) := \begin{cases} 1 & \text{if } \mathsf{w} \in \mathsf{inputs}(C) \\ \max \; \{b_{\mathsf{w},\mathsf{w}'} + \mathsf{depth}^\dagger(\mathsf{w}')\}_{\mathsf{w}'\in\mathsf{L\text{-}inputs}(g_{\mathsf{w}})} & \\ \quad \cup \{\mathsf{depth}^\dagger(\mathsf{w}')\}_{\mathsf{w}'\in\mathsf{R\text{-}inputs}(g_{\mathsf{w}})} & \text{if } \mathsf{w} \notin \mathsf{inputs}(C) \end{cases},$$
where
  - g_{w} (resp., $g_{\mathsf{w}'}$) is the gate in $\mathsf{gates}(C)$ for which $\mathsf{w} = \mathsf{output}(g_{\mathsf{w}})$ (resp., $\mathsf{w}' = \mathsf{output}(g_{\mathsf{w}'})$),
  - $b_{\mathsf{w},\mathsf{w}'} \in \{0,1\}$ equals 0 if and only if $\mathsf{R\text{-}deps}(g_{\mathsf{w}}) \supseteq \mathsf{R\text{-}deps}(g_{\mathsf{w}'})$.

Both $\star$-depth and $\dagger$-depth are bounded from above by (traditional) circuit depth, but are sometimes much less than it. For example, the circuit computing $\mathsf{w} \to (1, \mathsf{w}, \mathsf{w}^2, \ldots, \mathsf{w}^d)$ has depth d, while it has $\star$-depth and $\dagger$-depth equal to 1.

E. Pairings and duplex-pairing groups

Pairings. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be cyclic groups of a prime order r. Let $\mathcal{G}_1$ be a generator of $\mathbb{G}_1$, i.e., $\mathbb{G}_1 = \{\alpha\mathcal{G}_1\}_{\alpha\in\mathbb{F}_r}$, and let $\mathcal{G}_2$ be a generator for $\mathbb{G}_2$. A *pairing* is an efficient map $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_T$ is also a cyclic group of order r (which, unlike other groups, we write in multiplicative notation), satisfying the following properties.

- BILINEARITY. For every pair of nonzero elements $\alpha, \beta \in \mathbb{F}_r$, it holds that $e(\alpha\mathcal{G}_1, \beta\mathcal{G}_2) = e(\mathcal{G}_1, \mathcal{G}_2)^{\alpha\beta}$.
- NON-DEGENERACY. $e(\mathcal{G}_1, \mathcal{G}_2)$ is not the identity in $\mathbb{G}_T$.

Duplex-pairing groups. A group $\mathbb{G}$ of prime order r is *duplex pairing* if there are order-r groups $\mathbb{G}_1$ and $\mathbb{G}_2$ such that (i) there is a pairing $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for some target group $\mathbb{G}_T$, and (ii) there is a generator $\mathcal{G}_1$ of $\mathbb{G}_1$ and $\mathcal{G}_2$ of $\mathbb{G}_2$ such that $\mathbb{G}$ is isomorphic to $\{(t \cdot \mathcal{G}_1, t \cdot \mathcal{G}_2) | t \in \mathbb{F}_r\} \subseteq \mathbb{G}_1 \times \mathbb{G}_2$.

F. Multi-party broadcast protocols

We consider multi-party protocols that run over a synchronous network with an authenticated broadcast channel. Namely, the computation proceeds in rounds and, at each round, the protocol's schedule determines which parties act; a party acts by broadcasting a message to all other parties. The broadcast channel is authenticated in that all parties always know who sent a particular message (regardless of what an adversary may do). Moreover, we assume that parties have access to a common *random* string crs (the problem of "setting up" such a string via a multi-party protocol is well-studied). To simplify notation, we do not make crs an explicit input. We now introduce some notations and notions for later discussions.

Honest execution. Given a positive integer n, an n**-party broadcast protocol** is a tuple $\Pi = (S, \Sigma_1, \ldots, \Sigma_n)$ where: (i) $S \colon \mathbb{N} \to 2^{\{1,\ldots,n\}}$ is the deterministic polynomial-time *schedule* function; and (ii) for $i = 1, \ldots, n$, Σ_i is the (possibly stateful) probabilistic polynomial-time *strategy* of party i.

The *execution* of Π on an input $\vec{x} = (x_1, \ldots, x_n)$, denoted $[\![\Pi, \vec{x}]\!]$, works as follows. Set $t := 1$. While $S(t) \neq \emptyset$: (i) for each $i \in S(t)$ in any order, party i runs Σ_i, on input (x_i, t) and with oracle access to the history of messages broadcast so far, and broadcasts the resulting output message $\mathsf{msg}_{t,i}$ and, then, (ii) t increases by 1.

The *transcript* of $[\![\Pi, \vec{x}]\!]$, denoted tr, is the sequence of triples $(t, i, \mathsf{msg}_{t,i})$ ordered by $\mathsf{msg}_{t,i}$'s broadcast time. The *output* of $[\![\Pi, \vec{x}]\!]$, denoted out, is the last message in the transcript. Since Π's strategies are probabilistic, the transcript and output of $[\![\Pi, \vec{x}]\!]$ are random variables.

The *round complexity* is $\mathsf{ROUND}(\Pi) := \min_{t\in\mathbb{N}}\{t \,|\, S(t+1) = \emptyset\}$. For $i = 1, \ldots, n$, the *time complexity of party* i is $\mathsf{TIME}(\Pi, i) := \sum_{t\in[\mathsf{ROUND}(\Pi)] \text{ s.t. } i\in S(t)} \mathsf{TIME}(\Sigma_i, t)$ where $\mathsf{TIME}(\Sigma_i, t)$ is $\Sigma_i(\cdot, t)$'s time complexity.

Adversarial execution. Let A be a probabilistic polynomial-time algorithm and J a subset of $\{1, \ldots, n\}$. We denote by $[\![\Pi, \vec{x}]\!]_{A,J}$ the execution $[\![\Pi, \vec{x}]\!]$ modified so that A controls parties in J, i.e., A knows the private states of parties in J, may alter the strategies of parties in J, and may wait, in each round, to first see the messages broadcast by parties not in J and, only after that, instruct parties in J to send their messages. (In particular, $[\![\Pi, \vec{x}]\!]_{A,\emptyset} = [\![\Pi, \vec{x}]\!]$.) We denote by $\mathsf{REAL}_{\Pi,A,J}(\vec{x})$ the concatenation of the output of $[\![\Pi, \vec{x}]\!]_{A,J}$ and the view of A in $[\![\Pi, \vec{x}]\!]_{A,J}$.

G. Ideal functionalities

While Section III-F describes the real-world execution of a protocol Π on an input $\vec{x}$, here we describe the ideal-world execution of a function f on an input $\vec{x}$:

each party i privately sends his input x_i to a trusted party, who broadcasts $f(\vec{x})$.

Adversarial execution. Let S be a probabilistic polynomial-time algorithm and J a subset of $\{1,\dots,n\}$. The ideal-world execution of f on $\vec{x}$ when S controls parties in J differs from the above one as follows: S may substitute the inputs of parties in J with other same-length inputs. We denote by $\mathsf{IDEAL}_{f,S,J}(\vec{x})$ the concatenation of the value broadcast by the trusted party and the output of S in the ideal-world execution of f on $\vec{x}$ when S controls parties in J.

H. Secure sampling broadcast protocols

Let r be a prime, $\mathbb{G} = \langle \mathcal{G} \rangle$ an order-r group, n a positive integer, and $C\colon \mathbb{F}_r^m \to \mathbb{F}_r^h$ an $\mathbb{F}_r$-arithmetic circuit. A **secure sampling broadcast protocol with n parties for C over $\mathbb{G}$** is a tuple $\Pi^{\mathsf{SS}} = (\Pi, V, S)$, where Π is an n-party broadcast protocol, and V ((the *verifier*) and S (the *simulator*) are probabilistic polynomial-time algorithms, that satisfies the following.

For every probabilistic polynomial-time algorithm A (the *adversary*) and subset J of $\{1,\dots,n\}$ (the *corrupted parties*) with $|J| < n$, these two distributions are negligibly close:

$$\left\{ \mathsf{REAL}_{\Pi,A,J}(\vec{\sigma}) \;\middle|\; \begin{array}{c} \vec{\sigma}_1 \leftarrow \mathbb{F}_r^m \\ \vdots \\ \vec{\sigma}_n \leftarrow \mathbb{F}_r^m \end{array} \right\}_{V=1} \overset{\mathsf{negl}}{=} \left\{ \mathsf{IDEAL}_{f^{\mathsf{SS}}_{C,\mathcal{G}},S(A,J),J}(\vec{\sigma}) \;\middle|\; \begin{array}{c} \vec{\sigma}_1 \leftarrow \mathbb{F}_r^m \\ \vdots \\ \vec{\sigma}_n \leftarrow \mathbb{F}_r^m \end{array} \right\} .$$

Above, $\vec{\sigma}$ denotes $(\vec{\sigma}_1,\dots,\vec{\sigma}_n)$; $V = 1$ denotes conditioning on the event that V, given as input the transcript of $[\![\Pi,\vec{x}]\!]_{A,J}$, outputs 1; and $f^{\mathsf{SS}}_{C,\mathcal{G}}$denotes the deterministic function that outputs $C((\prod_{i=1}^n \sigma_{i,1},\dots,\prod_{i=1}^n \sigma_{i,m}))\cdot\mathcal{G}$, i.e., the encoding of C's output.

Next, we extend the above definition to variable number of parties and restricted circuit classes. Let r be a prime, $\mathbb{G} = \langle \mathcal{G} \rangle$ a group of order r, and $\mathbf{C}$ a class of $\mathbb{F}_r$-arithmetic circuits. A **secure sampling broadcast protocol for $\mathbf{C}$ over** $\mathbb{G}$ is a tuple $\Pi^{\mathsf{SS}} = (\Pi, V, S)$ such that, for every positive integer n and circuit $C\colon \mathbb{F}_r^m \to \mathbb{F}_r^h$ in $\mathbf{C}$, $(\Pi_{n,C}, V_{n,C}, S_{n,C})$ is a secure sampling broadcast protocol with n parties for C over $\mathbb{G}$.

I. Secure evaluation broadcast protocols

Let r be a prime, $\mathbb{G} = \langle \mathcal{G} \rangle$ an order-r group, n a positive integer, and $C\colon \mathbb{F}_r^{m_1}\times\cdots\times\mathbb{F}_r^{m_n} \to \mathbb{F}_r^h$ an $\mathbb{F}_r$-arithmetic circuit. A **secure evaluation broadcast protocol with n parties for C over** $\mathbb{G}$ is a tuple $\Pi^{\mathsf{SE}} = (\Pi, V, S)$, where Π is an n-party broadcast protocol and V, S are probabilistic polynomial-time algorithms, that satisfies the following.

For every probabilistic polynomial-time algorithm A, subset J of $\{1,\dots,n\}$ with $|J| < n$, and input $\vec{\sigma} = (\vec{\sigma}_1,\dots,\vec{\sigma}_n)$ in $\mathbb{F}_r^{m_1}\times\cdots\times\mathbb{F}_r^{m_n}$,

$$\{\mathsf{REAL}_{\Pi,A,J}(\vec{\sigma})\}_{V=1} \overset{\mathsf{negl}}{=} \left\{\mathsf{IDEAL}_{f^{\mathsf{SE}}_{C,\mathcal{G}},S(A,J),J}(\vec{\sigma})\right\} .$$

Above, $V = 1$ denotes the event that V, on input the transcript of $[\![\Pi,\vec{x}]\!]_{A,J}$, outputs 1, and $f^{\mathsf{SE}}_{C,\mathcal{G}}$ denotes the deterministic function such that $f^{\mathsf{SE}}_{C,\mathcal{G}}(\vec{\sigma}) := C(\vec{\sigma})\cdot\mathcal{G}$.

As before, we extend the above definition to variable number of parties and restricted circuit classes. Let r be a prime, $\mathbb{G} = \langle \mathcal{G} \rangle$ a group of order r, and $\mathbf{C}$ a class of $\mathbb{F}_r$-arithmetic circuits. A **secure evaluation broadcast protocol for $\mathbf{C}$ over** $\mathbb{G}$ is a tuple $\Pi^{\mathsf{SE}} = (\Pi, V, S)$ such that, for every positive integer n and circuit $C\colon \mathbb{F}_r^{m_1}\times\cdots\times\mathbb{F}_r^{m_n} \to \mathbb{F}_r^h$ in $\mathbf{C}$, $(\Pi_{n,C}, V_{n,C}, S_{n,C})$ is a secure evaluation broadcast protocol with n parties for C over $\mathbb{G}$.

IV. Secure sampling for a class of circuits

Our main construction is a multi-party protocol for securely sampling values of the form $C(\vec{\alpha})\cdot\mathcal{G}$ for a random $\vec{\alpha}$, provided that C belongs to the class $\mathbf{C}^\star$. We use two cryptographic ingredients: hiding commitments (see Section III-B) and NIZKs (see Section III-C). Recall that NIZKs require parties to access a common *random* string; such a string is available in the setting of multi-party broadcast protocols that we consider (see Section III-F). We prove the following theorem, by constructing the requisite protocol:

Theorem IV.1. *Assume the existence of hiding commitment schemes and NIZKs. Let r be a prime and $\mathbb{G}$ a group of order r. There is a secure sampling broadcast protocol $\Pi^{\mathsf{SS}} = (\Pi, V, S)$ for $\mathbf{C}^\star$ over $\mathbb{G}$ such that, for every positive integer n and circuit C in $\mathbf{C}^\star$:*

- *Round complexity:* $\mathsf{ROUND}(\Pi_{n,C}) = n\cdot\mathsf{depth}^\star(C) + 3$*;*
- *Time complexity: for $i = 1,\dots,n$,* $\mathsf{TIME}(\Pi_{n,C}, i) = O_\lambda(\mathsf{size}(C))$*; and*
- *Verification efficiency: $V_{n,C}$ runs in time $O_\lambda(n\cdot\mathsf{size}(C))$.*
- *Security (simulator efficiency): $S_{n,C}$ runs in time $O_\lambda(n\cdot\mathsf{size}(C))$.*

Concrete efficiency. It is crucial for our implementation that the use of NIZKs in the above theorem above is "light". We use NIZKs for two relations, denoted $\mathscr{R}_{\mathrm{A}}$ and $\mathscr{R}_{\mathrm{B}}$ and defined in Figure 1, that involve only

The NP **relation** $\mathscr{R}_{\mathrm{A}}$**.** An instance-witness pair $(\mathbb{x}, \mathbb{w})$ is in $\mathscr{R}_{\mathrm{A}}$ if and only if $\mathsf{COMM.Ver}(\sigma, \mathsf{cm}, \mathsf{trap}) = 1$, when parsing $\mathbb{x}$ as a commitment cm and $\mathbb{w}$ as a tuple (σ, trap) for which $\sigma \in \mathbb{F}_r$ and trap is a trapdoor.

The NP **relation** $\mathscr{R}_{\mathrm{B}}$. An instance-witness pair $(\mathbb{x}, \mathbb{w})$ is in $\mathscr{R}_{\mathrm{B}}$ if and only if all the following checks pass.

1) Parse $\mathbb{x}$ as tuple $(\mathcal{R}, \mathcal{P}, d, \vec{\alpha}, \vec{b}, \vec{c})$ and $\mathbb{w}$ as a tuple $(\vec{\sigma}, \vec{\mathsf{trap}})$.
2) Check that the $\mathbb{G}$-element $\mathcal{R}$ equals the $\mathbb{G}$-element $(\alpha_0 + \sum_{i=1}^{d} \alpha_i \sigma_i) \cdot \mathcal{P}$.
3) For $j = 1, \ldots, d$: if $b_j = 0$, check that $\mathsf{COMM.Ver}(\sigma_j, c_j, \mathsf{trap}_j) = 1$; otherwise, if $b_j = 1$, check that the $\mathbb{G}$-element c_j equals the $\mathbb{G}$-element $\sigma_j \cdot \mathcal{G}$ (and ignore trap_j).

Fig. 1. Description of the two NP relations $\mathscr{R}_{\mathrm{A}}$ and $\mathscr{R}_{\mathrm{B}}$.

arithmetic in $\mathbb{G}$ and invocations of the commitment verifier COMM.Ver. Indeed, if $\mathbb{G}$ is a duplex-pairing group (as when generating parameters for a zk-SNARK), it is possible to instantiate the commitment scheme, and subsequently the NIZKs for $\mathscr{R}_{\mathrm{A}}$ and $\mathscr{R}_{\mathrm{B}}$, very efficiently (see Section V). Our implementation and evaluation target this special case, as we discuss in later sections (see Sections VI and VII).

Proof strategy. We construct the protocol of Theorem IV.1 in two steps. The first step (Lemma IV.2 below) is a reduction from the problem of constructing secure *sampling* broadcast protocols to the problem of constructing secure *evaluation* broadcast protocols. The second step (Lemma IV.3) is a construction of such a secure evaluation broadcast protocol.

Lemma IV.2 (Sampling-to-evaluation reduction)**.** *Let r be a prime and $\mathbb{G}$ a group of order r. There exist polynomial-time transformations T_1 and T_2 that satisfy the following. For every positive integer n and circuit $C \in \mathbf{C}^\star$: $\tilde{C} := T_1(n, C)$ is a circuit in $\mathbf{C}^\dagger$, and for every secure evaluation broadcast protocol Π^{SE} with n parties for $\tilde{C}$ over $\mathbb{G}$, $\Pi^{\mathsf{SS}} := T_2(\Pi^{\mathsf{SE}})$ is a secure sampling broadcast protocol with n parties for C over $\mathbb{G}$.*

The transformation T_2 increases the protocol's round complexity by 1, and preserves all time complexities up to $O_\lambda(1)$ factors. Moreover, the new circuit $\tilde{C}$ is not much larger than C:

- $\mathsf{depth}^\dagger(\tilde{C}) = n \cdot \mathsf{depth}^\star(C)$*;*
- $\mathsf{size}(\tilde{C}) = O(n \cdot \mathsf{size}(C))$*; and*
- $\mathsf{size}(\tilde{C}, i) = O(\mathsf{size}(C))$ *for* $i = 1, \ldots, n$*.*

Lemma IV.3 (Evaluation protocol)**.** *Assume the existence of hiding commitment schemes and NIZKs. Let r be a prime and $\mathbb{G}$ a group of order r. There is a secure evaluation broadcast protocol $\Pi^{\mathsf{SE}} = (\Pi, V, S)$ for $\mathbf{C}^\dagger$ over $\mathbb{G}$ such that, for every positive integer n and circuit C in $\mathbf{C}^\dagger$:*

- $\mathsf{ROUND}(\Pi_{n,C}) = \mathsf{depth}^\dagger(C) + 2$*;*
- $\mathsf{TIME}(\Pi_{n,C}, i) = O_\lambda(\mathsf{size}(C, i))$ *for* $i = 1, \ldots, n$*; and*
- $V_{n,C}$ *and* $S_{n,C}$ *run in* $O_\lambda(\mathsf{size}(C))$ *time.*

Proofs of Lemma IV.2 and Lemma IV.3 are given in Appendix A and Appendix B, and sketched below.

A. *Sketch of Sampling-to-Evaluation Reduction*

We sketch the proof of Lemma IV.2 (see Appendix A for a detailed proof). At high level, the two transformations T_1 and T_2 work as follows. The circuit transformation T_1, given the number of parties n and a circuit $C \in \mathbf{C}^\star$, outputs a circuit $\tilde{C} \in \mathbf{C}^\dagger$ that computes C's output, along with other auxiliary values, by suitably combining n multiplicative shares of C's input. The protocol transformation T_2, given a secure evaluation protocol Π^{SE} for $\tilde{C}$, outputs a secure sampling protocol Π^{SS} for C by (i) generating random shares for all inputs, to ensure uniform sampling; (ii) extending the protocol by one round, to obtain a correctly-formatted output; (iii) extending the verifier, to account for the additional round in the transcript; and (iv) extending the simulator, to account for the different ideal functionality, whose output excludes the aforementioned auxiliary values (which, hence, must be simulated). Technically, most of the effort goes into constructing $\tilde{C}$ and the simulator of Π^{SS}. We thus briefly discuss these two.

The circuit $\tilde{C}$. We wish to have $\tilde{C}$ compute C's output from n multiplicative shares of C's input (which will be chosen at random). If this were the only requirement, then we could simply set $\tilde{C}$ equal to the circuit that, given as input n shares $\vec{\alpha}^{(1)}, \ldots, \vec{\alpha}^{(n)} \in \mathbb{F}^m$, first combines the share into $\vec{\alpha} := (\prod_{j=1}^{n} \alpha_1^{(j)}, \ldots, \prod_{j=1}^{n} \alpha_m^{(j)}) \in \mathbb{F}^m$ and then computes $C(\vec{\alpha})$. Unfortunately, such a circuit is not in the class $\mathbf{C}^\dagger$ (so cannot invoke Lemma IV.3 to securely evaluate $\tilde{C}$). The difficulty thus lies in constructing a circuit $\tilde{C}$ that computes the same function (perhaps with some additional, though simulatable, outputs) and that, moreover, is in $\mathbf{C}^\dagger$.

We thus take an alternative approach, which leverages the fact that C lies in $\mathbf{C}^\star$. For each gate g in C, we add to $\tilde{C}$ a sub-circuit, consisting of $O(n)$ new gates, that combines a value computed so far with all the shares of g's other input (which is, by definition of $\mathbf{C}^\star$, an input of C). Crucially, each of these sub-circuits, as well as their combination, lies in $\mathbf{C}^\dagger$. For example, suppose for simplicity that C actually has no gates that take more than one left input (i.e. all gates either multiply a constant by an input wire, or a previous output wire

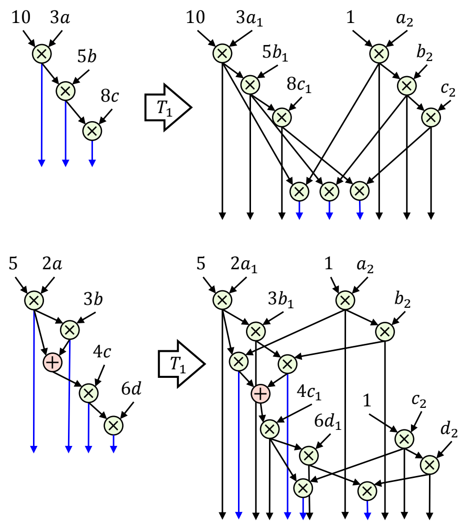

Fig. 2. Two examples of a circuit C in $\mathbf{C}^\star$ and the corresponding circuit $\tilde{C} := T_1(C, n)$ in $\mathbf{C}^\dagger$ for $n = 2$ parties. Multiplications are denoted by "$\times$", and additions by "$+$". The blue arrows in C denote the output wires of C; the blue arrows in $\tilde{C}$ denote the output wires of $\tilde{C}$ that compute outputs of C (while the remaining output wires carry partial computations).

by an input wire). Then $\tilde{C}$ contains n copies of C as a sub-circuit, each to be evaluated by a separate party, and corresponding gates' outputs, across the sub-circuits, are multiplied together by letting each party multiplicatively add a new share. See the first diagram in Figure 2.

However, when there are gates for which the left input is a non-trivial linear combination (i.e., has multiple terms), then the construction becomes somewhat more complex, because each party cannot anymore separately evaluate a copy of the circuit and rely on later gates to combine all the parties' evaluations. Roughly, the reason is that while multiplicative sharing commutes with "pure multiplications", they do not commute with additions. Therefore, we need to break the circuit down into components separated by linear combinations, and apply the above idea separately to each. In-between components each linear combination requires inputs that already aggregate all the parties' shares. See the second diagram in Figure 2 for an example.

A crucial efficiency feature of our reduction is that it ensures that the $\dagger$-depth of $\tilde{C}$, which determines the number of rounds required to securely evaluate $\tilde{C}$, is "small", i.e., bounded above by n times the $\star$-depth of C. Indeed, there are multiple ways to combine the aforementioned sub-circuits, but many such ways yield much worse efficiency, e.g., $\dagger$-depth that is as worse as n times the (standard) depth of C. Since the circuits C that we encounter in this paper's application have a small $\star$-depth, this efficiency is crucial.

The simulator in Π^{SS}. The construction of $\tilde{C}$ must not only respect syntactic and efficiency requirements (e.g., lie in $\mathbf{C}^\dagger$, not have more than $n \cdot \mathsf{size}(C)$ gates, and so on), but must also be secure, in the sense that the ideal functionality implemented by the evaluation protocol Π^{SE} for $\tilde{C}$ actually gives rise (with some small changes) to a sampling protocol Π^{SS} that implements the ideal functionality of C. Since our construction of $\tilde{C}$ introduces additional, spurious outputs, the simulator in Π^{SS} must be able to reproduce the view of the adversary when only having access to C's output (rather than $\tilde{C}$'s output). Intuitively, this requires showing that partial computations that carry information about a subset of the parties' shares (but not all shares) do not leak additional information beyond the outputs that incorporate every party's share.

For an arbitrary circuit in $\mathbf{C}^\dagger$ such an argument cannot be carried out. However, for the particular circuit $\tilde{C}$ that is constructed from C we show that it is possible to "back compute" the circuit: given the output of C, the simulator can complete it into an output of $\tilde{C}$ by sampling an assignment to the remaining (spurious) output wires of $\tilde{C}$, such that the simulated output is indistinguishable from an evaluation of $\tilde{C}$. This is done by taking each sub-circuit in $\tilde{C}$ and computing backwards from its output.

B. Sketch of Evaluation Protocol

We sketch the proof of Lemma IV.3 (see Appendix B for a detailed proof). The evaluation protocol $\Pi^{\mathsf{SE}} = (\Pi, V, S)$ proceeds as follows. In the first round, $(t = 1)$, each party i individually commits to each one of his own private inputs, i.e., each party i commits to the values assigned to wires in $\mathsf{inputs}(C, i)$, and proves, in zero knowledge, knowledge of the committed values (using relation $\mathscr{R}_{\mathrm{A}}$ of Figure 1). In each one of the subsequent $\mathsf{depth}^\dagger(C)$ rounds $(t = 2, \ldots, \mathsf{depth}^\dagger(C) + 1)$, each party i determines if there are any gates g in $\mathsf{gates}(C)$ such that the $\dagger$-depth of $\mathsf{output}(g)$ equals the round number minus 1 (i.e, $t - 1$) and $\mathsf{R\text{-}deps}(g) = \{i\}$; if so, then party i individually evaluates each such gate (in topological order) and broadcasts the result, along with a zero-knowledge proof that the evaluation was correct (using relation $\mathscr{R}_{\mathrm{B}}$ of Figure 1). In this way, the parties prove correct evaluation of all gates of C, first processing all gates whose outputs have $\dagger$-depth 1, then all those whose outputs have $\dagger$-depth 2, and so on. After $\mathsf{depth}^\dagger(C)$ such rounds, in the last round $(t = \mathsf{depth}^\dagger(C) + 2)$, party 1 consults the broadcast messages in order to collect, and then broadcast in a single message, the encoding of the value of every output wire of C.

(This only purpose of the last round is to construct a syntactically well-formed output of the protocol; letting party 1 do this operation is an arbitrary choice.)

Since $C \in \mathbf{C}^\dagger$, by definition of $\mathbf{C}^\dagger$, whenever a party is supposed to prove correct evaluation of a gate g, the left input has already been either computed by that party or broadcast; and the right input is either a constant or an input known to the party (since $\mathsf{R\text{-}deps}(g) = \{i\}$). This ensures that the party can evaluate the gate and knows an a witness to the NP statement about that gate's correct evaluation.

Also, since $C \in \mathbf{C}^\dagger$, every gate's output wire is also an output wire of C, so the broadcast of gate outputs do not leak information.

The transcript can be verified by checking that input commitments carry valid proofs and, for each gate, that the party responsible for that gate has produced valid proofs for its evaluation (based on suitable prior values); this ensures that the circuit has been evaluated on the parties' private inputs. Moreover, the transcript can be simulated, having access to the encoding of the circuit's output, by simulating each proof of correct evaluation.

V. Optimizations for duplex-pairing groups

While Theorem IV.1 holds for any choice of prime-order group $\mathbb{G}$, we obtain a particularly-efficient instantiation when $\mathbb{G}$ is a duplex-pairing group of order r. We describe this instantiation below.

Recall that the construction of the sampling protocol (Theorem IV.1) uses a reduction from sampling to evaluation (Lemma IV.2) and an evaluation protocol (Lemma IV.3). The reduction is quite efficient, so we focus on optimizing the construction of the evaluation algorithm, which uses commitments and NIZKs.

Choice of commitment scheme. We instantiate the commitment scheme COMM with Pedersen commitments [49]. Let $\mathcal{P}$ and $\mathcal{Q}$ be two generators of $\mathbb{G}$, for which there is no known linear relation (if $\mathbb{G}$ is an elliptic curve group then such $\mathcal{P}$ and $\mathcal{Q}$ can be found, for example, by applying point decompressing to two pseudorandom strings, e.g. SHA256(0) and SHA256(1)). A Pedersen commitment cm for a value x is obtained by letting trapdoor trap be a random element of $\mathbb{F}_r$ and computing $\mathsf{cm} := x \cdot \mathcal{P} + \mathsf{trap} \cdot \mathcal{Q}$. Recall that Pedersen commitments are statistically hiding (as $\mathsf{trap} \cdot \mathcal{Q}$ is uniformly random in $\mathbb{G}$) and computationally binding (as decommitting cm in two different ways yields a linear relation between $\mathcal{Q}$ and $\mathcal{P}$).

Choice of NIZK for the relation $\mathscr{R}_\mathbf{A}$. To prove knowledge of a commited value x encoded in a commitment cm, we use an adapted version of Schnorr's protocol [58] for zero-knowledge proof-of-knowledge of discrete logarithm. In the interactive version of the protocol, the prover first chooses random α and β in $\mathbb{F}_r$ and produces $\mathcal{R} = \alpha \cdot \mathcal{P} + \beta \cdot \mathcal{Q}$. The verifier responds with a uniformly sampled element c of $\mathbb{F}_r$, to which final prover message is $u := \alpha + c \cdot x$, $v := \beta + c \cdot \mathsf{trap}$. The verifier accepts iff $u \cdot \mathcal{P} + v \cdot \mathcal{Q} = \mathcal{R} + c \cdot \mathsf{cm}$. The protocol is made non-interactive by applying Fiat-Shamir heuristic [59] (in our concrete implementation, using SHA256 hashing).

Choice of NIZK for the relation $\mathscr{R}_\mathrm{B}$. We find that in our implementation we only need a special case of the relation $\mathscr{R}_\mathrm{B}$. For this special case what needs to be proved are the following two kinds of statements:

1) that a multiplicative relationship holds between a committed to value and two elements of $\mathbb{G}$: $(\mathcal{P}, \alpha, \mathcal{R}, 0, c) \in \mathscr{R}_\mathrm{B} \Leftrightarrow \mathcal{R} := \alpha\sigma \cdot \mathcal{P}$, where σ is equal to a value commited to in the commitment cm; and
2) that a multiplicative relationship holds between three elements of $\mathbb{G}$: $(\mathcal{P}, \alpha, \mathcal{R}, 1, c) \in \mathscr{R}_\mathrm{B} \Leftrightarrow \mathcal{R} := \alpha\sigma \cdot \mathcal{P}$, where $\sigma := \log_\mathcal{G} c$.

When $\mathbb{G}$ is a duplex pairing group, the proof for a statement of the second kind is just the empty string, as anyone can verify the statement by checking $e(\alpha\mathcal{P}, c) = e(\mathcal{R}, \mathcal{G})$.

To efficiently prove the statements of the first kind, we slightly modify the construction of Lemma IV.3. We insert an additional round after the first round (in which all parties commit to their inputs). In this additional round each party, for each of its inputs x samples a random generator $\mathcal{P}$ of $\mathbb{G}$, computes $\mathcal{R} := x \cdot \mathcal{P}$ and outputs $(\mathcal{P}, \mathcal{R})$. Moreover, the party outputs a NIZK proof-of-knowledge that the implicitly defined $\hat{x} := \log_\mathcal{P} \mathcal{R}$ is indeed consistent with the corresponding commitment cm, i.e. cm for x decommits to $\log_\mathcal{P} \mathcal{R}$. Call the corresponding relation $\mathscr{R}_\mathsf{aux}$. Note that, publishing such encodings $(\mathcal{P}, x \cdot \mathcal{P})$ of inputs x does not break confidentiality: a pair $(\mathcal{Q}, x \cdot \mathcal{Q})$ (for some $\mathcal{Q}$) is necessarily output every time an input x is used in a multiplication gate. By a hybrid argument, having polynomially many such pairs is as helpful as having just one. Equipped with such encodings checking $\mathscr{R}_\mathrm{B}$ can be done just via pairing evaluations.

Finally, a NIZK proof for relation $\mathscr{R}_\mathsf{aux}$ is obtained by combining the Σ-protocol for knowledge of a Pedersen commitment, and Schnorr's Σ-protocol for knowledge of discrete logarithm in equality composition [60]. As above, we make the resulting Σ-protocol non-interactive by applying Fiat-Shamir heuristic.

VI. Implementation

Our system. We built a system that implements our constructions. Given a prime r, an order-r duplex-pairing group $\mathbb{G} = \langle \mathcal{G} \rangle$, and an $\mathbb{F}_r$-arithmetic circuit $C \colon \mathbb{F}_r^m \to \mathbb{F}_r^h$ in the class $\mathbf{C}^\star$, our system provides a multi-party protocol for securely sampling $C(\vec{\alpha}) \cdot \mathcal{G}$ for random $\vec{\alpha}$ in $\mathbb{F}_r^m$. Specifically, the system implements the constructions underlying Section IV's theorems, in the case when $\mathbb{G}$ is a duplex-pairing group. (As discussed in Section V, if $\mathbb{G}$ is duplex-pairing, one can instantiate hiding commitments and NIZKs very efficiently.) Our system comprises 1141 lines of `C++`.

Application to zk-SNARKs via integration with `libsnark`. The parameter generator of many zk-SNARK constructions works as follows: evaluate a certain circuit C at a random input $\vec{\alpha}$, and then output $\mathsf{pp} := C(\vec{\alpha}) \cdot \mathcal{G}$ as the proof system's public parameters. (See discussion in Section I-A.) Thus, our system can be used to securely sample public parameters of a zk-SNARK, provided that the circuit used in its generator belongs to the circuit class $\mathbf{C}^\star$. To facilitate this application, we have integrated our code with `libsnark` [46], a `C++` library for zk-SNARKs. (In particular, the sampled pp can be used directly by `libsnark`.)

Two zk-SNARK constructions. We worked out circuits for parameter generation for two (preprocessing) zk-SNARK constructions: the one of [21], [25] and the one of [31]. The first zk-SNARK "natively" supports proving satisfiability of *arithmetic* circuits, while the second zk-SNARK that of *boolean* circuits.[8]

Specifically, we wrote code that lays out a circuit $C_{\mathsf{PGHR}} \in \mathbf{C}^\star$ that can be used to generate public parameters for [21], [25]'s zk-SNARK; likewise for laying out a circuit $C_{\mathsf{DFGK}} \in \mathbf{C}^\star$ for [31]'s zk-SNARK. We have invoked our system on both circuit types, and demonstrated the secure sampling of respective public parameters.

See Appendix C for more information about these examples. A critical issue discussed there is ensuring that C_{PGHR} and C_{DFGK} have size quasilinear in the circuit whose satisfiability is being proved. A naive imitation of the computation pattern of the zk-SNARK's generator results in circuits that are not in $\mathbf{C}^\star$; conversely, a naive implementation in $\mathbf{C}^\star$ results in circuits of quadratic size. Via careful design, quasilinear-size circuits in $\mathbf{C}^\star$ can be obtained.

[8]More precisely, both [21], [25] and [31] actually support more general NP relations (phrased in terms of systems of equations), but we ignore this technical detail in this and later discussions.

VII. Evaluation

We describe the evaluation of our system, which provides a multi-party protocol for securely sampling $C(\vec{\alpha}) \cdot \mathcal{G}$, where $\vec{\alpha}$ is random, for circuits C that belong to the circuit class $\mathbf{C}^\star$ (see Section VI).

Setup. We evaluated our system on a desktop PC with a 3.40 GHz Intel Core i7-4770 CPU and 16 GB of RAM available. All experiments are in single-thread mode (though our code also supports multiple-thread mode). When invoking functionality from `libsnark` (with which our code is integrated), we selected the build option `CURVE=BN128`, which means that group arithmetic is conducted over a certain Barreto–Naehrig curve [61] at 128 bits of security.

Costs for the general case. Our system's efficiency only depends on the size and $\star$-depth of the circuit C in $\mathbf{C}^\star$, and also n (the number of participating parties). In Figure 4 we report approximate costs for several complexity measures: the number of rounds, each party's time complexity, the number of broadcast messages, the transcript size, and the transcript verification time. (Figure 4 also displays costs for two concrete examples, discussed further down below.)

Costs for two zk-SNARK constructions. When applying our system to generate public parameters for a zk-SNARK, the circuit C is designed so that $C(\vec{\alpha}) \cdot \mathcal{G}$ (for random $\vec{\alpha}$) equals the zk-SNARK's generator output distribution. This distribution depends on the particular NP relation given as input to the generator; thus, the circuit C also depends on this NP relation. Moreover, different zk-SNARK constructions "natively" support different classes of NP relations.

In order to shed light on our system's efficiency when applied to generate zk-SNARK public parameters, we report the size and $\star$-depth of the circuit C as a function of the input NP relation, relative to two zk-SNARK constructions.

- *The zk-SNARK [31].* This zk-SNARK supports boolean circuit satisfiability: the generator receives as input a boolean circuit D, and outputs public parameters for proving D's satisfiability. If D has N_{w} wires and N_{g} gates, our code outputs a corresponding circuit $C := C_{\mathsf{DFGK}}$ with size $2 \cdot N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 10$ and $\star$-depth 2.
- *The zk-SNARK of [21], [25].* This zk-SNARK supports arithmetic circuit satisfiability: the generator receives as input an arithmetic circuit D, and outputs public parameters for proving D's satisfiability. If D has N_{w} wires and N_{g} gates, our code outputs a circuit $C := C_{\mathsf{PGHR}}$ with size $11 \cdot N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 38$ and $\star$-depth 3.

In Figure 3 we summarize these costs, alongside costs for two concrete examples, which we discuss next.

Costs for two concrete examples. We also report costs for the following concrete choices of a circuit $C := C_{\mathsf{PGHR}}$.

- **Example #1:** the circuit C targets Zerocash [44]. Namely, $C(\vec{\alpha}) \cdot \mathcal{G}$ (for random $\vec{\alpha}$) equals the output distribution of the generator of the preprocessing zk-SNARK on which Zerocash is based. We selected this example because [44] 's authors had acknowledged the need, in practice, to securely sample Zerocash's parameters.
- **Example #2:** the circuit C targets the scalable zk-SNARK of [39]. Namely, $C(\vec{\alpha}) \cdot \mathcal{G}$ (for random $\vec{\alpha}$) equals the output distribution of the generator used to set up the scalable zk-SNARK. We selected this example because, in this case, the generator's output is *universal* (it suffices for proving *any* computation expressed as machine code on a certain RISC machine), so that the zk-SNARK's parameters can be securely sampled once and for all.

In Figure 3 we report the size and $\star$-depth of C for these two examples, and in Figure 4 we report the corresponding costs of our system when run on these choices of C.

VIII. Conclusion

Like time and space, trust is also a costly resource. To facilitate the deployment of NIZKs and, in particular, zk-SNARKs in various applications, it is not only important to minimize the time and space requirements of proving and verification, but also the trust requirements of parameter generation.

The system that we have presented in this paper can be used to reduce the trust requirements of parameter generation for a class of zk-SNARKs: the system provides a multi-party broadcast protocol in which only one honest party, out of n participating ones, is required to securely sample the public parameters. Integration of our system with `libsnark` greatly facilitates this application. As a demonstration, we have used our system for securely sampling public parameters for the zk-SNARKs of [21], [25], [31].

It is an interesting question to extend our system so to support circuits C that are not in the circuit class $\mathbf{C}^{\star}$. Can use continue to rely on relatively "light" cryptographic techniques? Such a system would be able to support parameter generation of essentially all preprocessing zk-SNARKs.

Finally, in this work we have not attempted to tackle the "human component" of parameter generation. Namely, once we have a system that allows secure sampling via a multi-party protocol, how should we choose the participating parties? What penalties should be put in place for misbehavior, if any? Where and how should the protocol be conducted? These questions, too, need good answers in order to convincingly sample public parameters via the multi-party protocol.

Appendix A
Proof of Lemma IV.2

We prove Lemma IV.2. Specifically, first we describe the construction of the circuit transformation T_1, and then the construction of the protocol transformation T_2; afterwards, we explain why these constructions work.

Construction of $\boldsymbol{T_1}$. On input a positive integer n and a circuit $C\colon \mathbb{F}_r^m \to \mathbb{F}_r^h$ in the class $\mathbf{C}^{\star}$, the transformation T_1 outputs a circuit $\tilde{C}$ in the class $\mathbf{C}^{\dagger}$.

First we describe high-level properties of the circuit $\tilde{C}$. The number of wires, gates, inputs, and outputs of $\tilde{C}$ is a multiplicative factor of n larger than those of C: $\#\mathsf{wires}(\tilde{C}) = n \cdot \#\mathsf{wires}(C)$, $\#\mathsf{gates}(\tilde{C}) = n \cdot \#\mathsf{gates}(C)$, $\#\mathsf{inputs}(\tilde{C}) = n \cdot \#\mathsf{inputs}(C)$, and $\#\mathsf{outputs}(\tilde{C}) = n \cdot \#\mathsf{outputs}(C)$. The inputs of $\tilde{C}$ are partitioned into n disjoint sets each of size m and, for each i, $\mathsf{size}(\tilde{C}, i) = O(\mathsf{size}(C))$. In particular, we can write $\tilde{C}\colon \mathbb{F}^{m_1} \times \cdots \times \mathbb{F}^{m_n} \to \mathbb{F}^{nh}$ with each m_i equal to m. The $\dagger$-depth of $\tilde{C}$ is n times larger than the $\star$-depth of C: $\mathsf{depth}^{\dagger}(\tilde{C}) = n \cdot \mathsf{depth}^{\star}(C)$.

Moreover, there is a *wire embedding* from C to $\tilde{C}$, i.e., a map $\phi\colon \mathsf{outputs}(C) \to \mathsf{outputs}(\tilde{C})$ that works as follows. Consider any $\vec{\alpha}^{(1)}, \dots, \vec{\alpha}^{(n)} \in \mathbb{F}^m$ and let $\vec{\alpha} := (\prod_{j=1}^{n} \alpha_1^{(j)}, \dots, \prod_{j=1}^{n} \alpha_m^{(j)}) \in \mathbb{F}^m$. Then, for every output wire w of C, the value assigned to w when computing $C(\vec{\alpha})$ equals the value assigned to $\phi(\mathsf{w})$ when computing $\tilde{C}(\vec{\alpha}^{(1)}, \dots, \vec{\alpha}^{(n)})$. In other words, if $\tilde{C}$'s input corresponds to a multiplicative sharing, among n parties, of C's input, then $\tilde{C}$'s output contains C's output (as well as other values), and ϕ specifies the embedding from the latter into the former. In particular, if each of the shares are non-zero and at least one party's shares are all random, then the distribution of $\tilde{C}$'s output coincides with the distribution obtained by evaluating C at a random input (having no zeros).

We now turn to the construction of the circuit $\tilde{C}$ from C. We assume, for notational convenience, that C's wires have an order: for every wire $\mathsf{w} \in \mathsf{wires}(C)$, $\mathsf{idx}(\mathsf{w})$ denotes the wire's *index* in $\{1, \dots, \#\mathsf{wires}(C)\}$ of w according to this order; moreover, C's input wires have indexes from 1 to m. Initialize $\tilde{C}$ to be a gate-less circuit with domain $\mathbb{F}^{m_1} \times \cdots \times \mathbb{F}^{m_n}$ (recall the definition of circuits with a split domain, in Section III-D), and denote by $\tilde{\mathsf{w}}_{i,k}^{\mathsf{in}}$ the i-th input wire of $\tilde{C}$'s k-th input set (here i ranges from 1 to m, while k ranges from 1 to

zk-SNARK	**Circuit satisfiability of D when D is**	**Circuit C in $\mathbf{C}^\star$**	
		$\mathsf{size}(C)$	$\mathsf{depth}^\star(C)$
Danezis et al. [31]	a N_w-wire N_g-gate boolean circuit	$2 \cdot N_\mathsf{w} + 2^{\lceil \log_2 N_\mathsf{g} \rceil}(\lceil \log_2 N_\mathsf{g} \rceil + 1) + 10$	2
Parno et al., Ben-Sasson et al. [21], [25]	a N_w-wire N_g-gate arithmetic circuit	$11 \cdot N_\mathsf{w} + 2^{\lceil \log_2 N_\mathsf{g} \rceil}(\lceil \log_2 N_\mathsf{g} \rceil + 1) + 38$	3
Ben-Sasson et al. [44]	Example #1's arithmetic circuit	138467206	3
Ben-Sasson et al. [39]	Example #2's arithmetic circuit	8027609	6

Fig. 3. Size and $\star$-depth of the circuit C in $\mathbf{C}^\star$ obtained from D, for various choices of D.

Complexity measure	**Cost for**		
	general case	**Example #1**	**Example #2**
number of rounds	$n \cdot \mathsf{depth}^\star(C) + 3$	$3 \cdot n + 3$	$6 \cdot n + 6$
each party's time complexity	$0.035 \cdot \mathsf{size}(C)$ ms	14124 s	4048 s
number of broadcast messages	$n \cdot (\mathsf{depth}^\star(C) + 3)$	$6 \cdot n$	$6 \cdot n$
transcript size	$0.072 \cdot n \cdot \mathsf{size}(C)$ kB	$12877 \cdot n$ MB	$906 \cdot n$ MB
transcript verification time	$1.03 \cdot n \cdot \mathsf{size}(C)$ ms	$196208 \cdot n$ s	$50945 \cdot n$ s

Fig. 4. Our system's costs for the general case, Example #1, and Example #2.

n). The procedure described below iteratively adds gates and wires to $\tilde{C}$ by considering in turn each wire of C. It also builds an $\#\mathsf{outputs}(C)$-size vector $\vec{\phi}$ representing the wire embedding ϕ

For each non-input wire w of C (i.e., in $\mathsf{wires}(C) \setminus \mathsf{inputs}(C)$) taken in topological order, do the following.

1) Denote $i = \mathsf{idx}(\mathsf{w})$. Let g be the gate in $\mathsf{gates}(C)$ that computes $(\alpha_0^\mathsf{L} + \sum_{j=1}^{d^\mathsf{L}} \alpha_j^\mathsf{L} \mathsf{w}_j^\mathsf{L}) \cdot (\alpha_0^\mathsf{R} + \sum_{j=1}^{d^\mathsf{R}} \alpha_j^\mathsf{R} \mathsf{w}_j^\mathsf{R}) \to \mathsf{w}$. Note that as $C \in \mathbf{C}^\star$, we have $d^\mathsf{R} \leq 1$ and $\alpha_0^\mathsf{R} = 0$ if $d^\mathsf{R} = 1$.
2) Distinguish between several cases, depending on g:
   - $d^\mathsf{L} = 0 \wedge d^\mathsf{R} = 1$ *(i.e., g is a "constant-times-input" gate).*
     a) Set $r := \mathsf{idx}(\mathsf{w}_1^\mathsf{R})$.
     b) Add n new wires $\tilde{\mathsf{w}}_{i,1}, \ldots, \tilde{\mathsf{w}}_{i,n}$ to $\tilde{C}$, and also n new gates $\tilde{g}_1, \ldots, \tilde{g}_n$.
     c) Have $\tilde{g}_1$ compute $\alpha_0^\mathsf{L} \cdot (\alpha_1^\mathsf{R} \tilde{\mathsf{w}}_{r,1}^\mathsf{in}) \to \tilde{\mathsf{w}}_{i,1}$ and for $k = 2, \ldots, n$ have $\tilde{g}_k$ compute $1 \cdot \tilde{\mathsf{w}}_{r,k}^\mathsf{in} \to \tilde{\mathsf{w}}_{i,k}$.
     d) Add $n - 1$ new wires $\tilde{\mathsf{w}}'_{i,1}, \ldots, \tilde{\mathsf{w}}'_{i,n-1}$ to $\tilde{C}$, and also $n - 1$ new gates $\tilde{g}'_{i,1}, \ldots, \tilde{g}'_{i,n-1}$.
     e) Have $\tilde{g}'_{i,1}$ compute $\tilde{\mathsf{w}}_{i,1} \cdot \tilde{\mathsf{w}}_{i,2} \to \tilde{\mathsf{w}}'_{i,1}$ and for $k = 2, \ldots, n - 1$ have $\tilde{g}'_{i,k}$ compute $\tilde{\mathsf{w}}'_{i,k-1} \cdot \tilde{\mathsf{w}}_{i,k} \to \tilde{\mathsf{w}}'_{i,k}$.
   - $(d^\mathsf{L} = 1 \wedge \alpha_0^\mathsf{L} = 0) \wedge d^\mathsf{R} = 1$ *(i.e., g is an "output-times-input" gate).*
     a) Set $l := \mathsf{idx}(\mathsf{w}_1^\mathsf{L})$ and $r := \mathsf{idx}(\mathsf{w}_1^\mathsf{R})$; let g_l be the gate in $\mathsf{gates}(C)$ that computes w_l with $\mathsf{idx}(\mathsf{w}_l) = l$.
     b) Add n new wires $\tilde{\mathsf{w}}_{i,1}, \ldots, \tilde{\mathsf{w}}_{i,n}$ to $\tilde{C}$, and also n new gates $\tilde{g}_1, \ldots, \tilde{g}_n$.
     c) If $|\mathsf{L\text{-}inputs}(g_l)| = 1$ and $\mathsf{L\text{-}coeffs}(g_l)[0] = 0$, then have $\tilde{g}_1$ compute $(\alpha_1^\mathsf{L} \tilde{\mathsf{w}}_{l,1}) \cdot (\alpha_1^\mathsf{R} \tilde{\mathsf{w}}_{r,1}^\mathsf{in}) \to \tilde{\mathsf{w}}_{i,1}$ and for $k = 2, \ldots, n$ have $\tilde{g}_k$ compute $\tilde{\mathsf{w}}_{l,k} \cdot \tilde{\mathsf{w}}_{r,k}^\mathsf{in} \to \tilde{\mathsf{w}}_{i,k}$.
     d) If $|\mathsf{L\text{-}inputs}(g_l)| > 1$ or $\mathsf{L\text{-}coeffs}(g_l)[0] \neq 0$, then have $\tilde{g}_1$ compute $(\alpha_1^\mathsf{L} \tilde{\mathsf{w}}_{l,n}) \cdot (\alpha_1^\mathsf{R} \tilde{\mathsf{w}}_{r,1}^\mathsf{in}) \to \tilde{\mathsf{w}}_{i,1}$ and for $k = 2, \ldots, n$ have $\tilde{g}_k$ compute $1 \cdot \tilde{\mathsf{w}}_{r,k}^\mathsf{in} \to \tilde{\mathsf{w}}_{i,k}$.
     e) Add $n - 1$ new wires $\tilde{\mathsf{w}}'_{i,1}, \ldots, \tilde{\mathsf{w}}'_{i,n-1}$ to $\tilde{C}$, and also $n - 1$ new gates $\tilde{g}'_{i,1}, \ldots, \tilde{g}'_{i,n-1}$.
     f) Have $\tilde{g}'_{i,1}$ compute $\tilde{\mathsf{w}}_{i,1} \cdot \tilde{\mathsf{w}}_{i,2} \to \tilde{\mathsf{w}}'_{i,1}$ and for $k = 2, \ldots, n - 1$ have $\tilde{g}'_{i,k}$ compute $\tilde{\mathsf{w}}'_{i,k-1} \cdot \tilde{\mathsf{w}}_{i,k} \to \tilde{\mathsf{w}}'_{i,k}$.
   - $\big(d^\mathsf{L} \geq 2 \vee (d^\mathsf{L} = 1 \wedge \alpha_0^\mathsf{L} \neq 0)\big) \wedge d^\mathsf{R} = 1$ *(i.e., g is a "linear-combination-times-input" gate).*
     a) Set $l_1 := \mathsf{idx}(\mathsf{w}_1^\mathsf{L}), \ldots, l_{d^\mathsf{L}} := \mathsf{idx}(\mathsf{w}_{d^\mathsf{L}}^\mathsf{L})$.
     b) Add n new wires $\tilde{\mathsf{w}}_{i,1}, \ldots, \tilde{\mathsf{w}}_{i,n}$ to $\tilde{C}$, and also n new gates $\tilde{g}_1, \ldots, \tilde{g}_n$.
     c) Have $\tilde{g}_1$ compute $(\alpha_0^\mathsf{L} + \sum_{j=1}^{d^\mathsf{L}} \alpha_j^\mathsf{L} \tilde{\mathsf{w}}'_{j,n-1}) \cdot (\alpha_1^\mathsf{R} \tilde{\mathsf{w}}_{r,k}^\mathsf{in}) \to \tilde{\mathsf{w}}_{i,1}$ and for $k = 2, \ldots, n$ have $\tilde{g}_k$ compute $\tilde{\mathsf{w}}_{l,k} \cdot \tilde{\mathsf{w}}_{r,k}^\mathsf{in} \to \tilde{\mathsf{w}}_{i,k}$.
     d) Henceforth for $k = 1, \ldots, n - 1$ treat all references to $\tilde{\mathsf{w}}'_{i,k}$ as a reference to the single wire $\tilde{\mathsf{w}}_i$.
   - $d^\mathsf{L} = 0 \wedge d^\mathsf{R} = 0$ *(i.e., g is "constant-times-constant" gate).*
     a) Add a new wire $\tilde{\mathsf{w}}_i$ to $\tilde{C}$, and a new gate $\tilde{g}$.
     b) Have $\tilde{g}$ compute $\alpha_0^\mathsf{L} \cdot \alpha_0^\mathsf{R} \to \tilde{\mathsf{w}}_i$.
     c) Henceforth for $k = 1, \ldots, n - 1$ treat all references to $\tilde{\mathsf{w}}'_{i,k}$ as a reference to the single wire $\tilde{\mathsf{w}}_i$.
   - $(d^\mathsf{L} = 1 \wedge \alpha_0^\mathsf{L} = 0) \wedge d^\mathsf{R} = 0$ *(i.e., g is an "output-times-constant" gate).*
     a) Set $l := \mathsf{idx}(\mathsf{w}_1^\mathsf{L})$.
     b) Add a new wire $\tilde{\mathsf{w}}_i$ to $\tilde{C}$, and a new gate $\tilde{g}$.
     c) Have $\tilde{g}$ compute $(\alpha_1^\mathsf{L} \tilde{\mathsf{w}}'_{l,n-1}) \cdot \alpha_0^\mathsf{R} \to \tilde{\mathsf{w}}_i$.

d) Henceforth for $k = 1, \ldots, n-1$ treat all references to $\tilde{\mathsf{w}}'_{i,k}$ as a reference to the single wire $\tilde{\mathsf{w}}_i$.

- $\big(d^{\mathsf{L}} \geq 2 \;\vee\; (d^{\mathsf{L}} = 1 \;\wedge\; \alpha_0^{\mathsf{L}} \neq 0)\big) \;\wedge\; d^{\mathsf{R}} = 0$ *(g is a "linear-combination-times-constant" gate).*
  a) Set $l_1 := \mathsf{idx}(\mathsf{w}_1^{\mathsf{L}}), \ldots, l_{d^{\mathsf{L}}} := \mathsf{idx}(\mathsf{w}_{d^{\mathsf{L}}}^{\mathsf{L}})$.
  b) Add a new wire $\tilde{\mathsf{w}}_i$ to $\tilde{C}$, and a new gate $\tilde{g}$.
  c) Have $\tilde{g}$ compute $(\alpha_0^{\mathsf{L}} + \sum_{j=1}^{d^{\mathsf{L}}} \alpha_j^{\mathsf{L}} \tilde{\mathsf{w}}'_{j,n-1}) \cdot \alpha_0^{\mathsf{R}} \rightarrow \tilde{\mathsf{w}}_{i,k}$.
  d) Henceforth for $k = 1, \ldots, n-1$ treat all references to $\tilde{\mathsf{w}}'_{i,k}$ as a reference to the single wire $\tilde{\mathsf{w}}_i$.

3) Set $\vec{\phi}[i] := \mathsf{idx}(\tilde{\mathsf{w}}'_{i,n-1})$.
4) Require that outputs of gates added to $\tilde{C}$ are also outputs of $\tilde{C}$.

Note that, above, for each gate of C, we add $O(n)$ gates and $O(n)$ wires to $\tilde{C}$; hence, $\mathsf{size}(\tilde{C}) = O(n \cdot \mathsf{size}(C))$. Moreover, the gates of $\tilde{C}$ that reference $\tilde{\mathsf{w}}^{\mathsf{in}}$ correspond to gates of C that a reference w^{in}; if a gate g of C references an input wire w^{in}, then the corresponding n gates of $\tilde{C}$ will reference one wire from each parties' shares; hence, $\mathsf{size}(\tilde{C}) = O(n \cdot \mathsf{size}(C))$ and $\mathsf{size}(\tilde{C}, i) = O(\mathsf{size}(C))$ for $i = 1, \ldots, n$. Finally, it's easy to check that $\mathsf{depth}^{\dagger}(\tilde{C}) = n \cdot \mathsf{depth}^{\star}(C)$.

Construction of T_2. On input a secure evaluation broadcast protocol $\Pi^{\mathsf{SE}} = (\Pi, V, S)$ with n parties for $\tilde{C}$, the transformation T_2 outputs a triple $\Pi^{\mathsf{SS}} = (\Pi', V', S')$ that is constructed as follows. (Allegedly, Π^{SS} is a secure sampling broadcast protocol with n parties for C.)

- *Construction of* Π'. Let $\Pi = (S, \Sigma_1, \ldots, \Sigma_n)$. Construct $\Pi' := (S', \Sigma'_1, \ldots, \Sigma'_n)$ as follows. The schedule S' is

$$S'(t) := \begin{cases} S(t) & \text{if } 0 < t \leq \mathsf{ROUND}(\Pi) \\ \{1\} & \text{if } t = \mathsf{ROUND}(\Pi) + 1 \\ \emptyset & \text{otherwise} \end{cases} .$$

  Next, for $i = 1, \ldots, n$, the strategy Σ'_i, on input (x_i, t) and with oracle access to the history of messages broadcast so far, works as follows.
  - If $0 < t \leq \mathsf{ROUND}(\Pi)$, do the following. Run Σ_i (on the same inputs received by Σ'_i) and output its output message $\mathsf{msg}_{t,i}$.
  - If $t = \mathsf{ROUND}(\Pi) + 1$ and $i = 1$, do the following. Collect the encoding of the value of every output wire of C. This can be done by using the wire embedding ϕ to select values from the last message broadcast so far because, by definition (being the last message broadcast in Π), this message contains the encoding of the value of every output wire of $\tilde{C}$. Set $\mathsf{msg}_{t,i}$ equal to the vector of these selected encodings and output $\mathsf{msg}_{t,i}$.

  Namely, the first $\mathsf{ROUND}(\Pi)$ rounds of Π' coincide with the first $\mathsf{ROUND}(\Pi)$ rounds of Π. Then, in the last round of Π', party 1 (chosen arbitrarily) collects from the output of Π the encodings needed to create the output for Π'.
- *Construction of* V'. On input a transcript tr, the verifier V' works as follows. Let $\tilde{\mathsf{msg}}$ denote the last message in tr, and $\tilde{\mathsf{tr}}$ the transcript obtained by removing $\tilde{\mathsf{msg}}$ from tr. Check that $V(\tilde{\mathsf{tr}}) = 1$. Then check that $\tilde{\mathsf{msg}}$ equals the message obtained by using ϕ to select from the last message in $\tilde{\mathsf{tr}}$ the outputs of C.
- *Construction of* S'. On input an adversary A and set J of corrupted parties, the simulator S' works as follows. (We assume that $|J| > 0$, for otherwise the simulation is trivial.)
  1) *Construct a new adversary* $\tilde{A}$. The simulator first modifies the adversary A, which is an adversary against the sampling protocol $\Pi^{\mathsf{SS}} = (\Pi', V', S')$, to an adversary $\tilde{A}$ against the evaluation protocol $\Pi^{\mathsf{SE}} = (\Pi, V, S)$. By construction of Π', this can be done by designing $\tilde{A}$ so that (i) $\tilde{A}$ runs A and lets it interact with the outside world up to and including round $\mathsf{ROUND}(\Pi)$ (up to this round, Π' and Π are identical); and (ii) $\tilde{A}$ simulates for A the last round (the only round at which Π' and Π differ). The last round can be easily simulated by $\tilde{A}$ because, consisting merely of collecting some values from past messages, it is a public operation on the view of A.
  2) *Run* Π^{SE}*'s simulator on the new adversary* $\tilde{A}$. The simulator runs S on input the new adversary $\tilde{A}$ and the set J of corrupted parties. When S outputs, for each $i \in J$, the (extracted) malicious input $\vec{\sigma_i}^* := (\sigma^*_{i,j})_{j=1}^{m_i}$ for party i, the simulator forwards it to the trusted party.
  At the same time, each honest party $i \in J$ sends to the trusted party his own vector $\sigma_i := (\sigma_{i,j})_{j=1}^{m_i}$. The trusted party, broadcasts the output $f^{\mathsf{SS}}_{C,\mathcal{G}}(\vec{\sigma}^*)$, where $\vec{\sigma}^*$ combines $(\vec{\sigma_i}^*)_{i \in J}$ and $(\vec{\sigma_i})_{i \notin J}$ in order of i. Note that each malicious input $\vec{\sigma_i}^*$ was not intended as an input to the function $f^{\mathsf{SS}}_{C,\mathcal{G}}$, but instead to $f^{\mathsf{SE}}_{\tilde{C},\mathcal{G}}$. Though, while different, the two functions have the same domain, and thus the trusted party's output is well-defined.
  Next, the simulator must relay to S a value for $f^{\mathsf{SE}}_{\tilde{C},\mathcal{G}}$, while only having access to the trusted party's output, $f^{\mathsf{SS}}_{C,\mathcal{G}}(\vec{\sigma}^*) := C\big((\prod_{i=1}^n \sigma^*_{i,1}, \ldots, \prod_{i=1}^n \sigma^*_{i,m})\big) \cdot \mathcal{G}$, and the (extracted) malicious inputs, $(\vec{\sigma_i}^*)_{i \in J}$. Crucially, the relayed value must be indistinguishable from the value that S would have seen if S

had accessed the function $f^{\mathsf{SE}}_{\tilde{C},\mathcal{G}}$. This particular simulation is the core of the simulator and is discussed separately in the next step.

3) *Simulation of* $f^{\mathsf{SE}}_{\tilde{C},\mathcal{G}}(\vec{\sigma}^*)$. Let $\vec{\mathcal{X}}$ be the distribution over $\mathbb{F}^{m_1} \times \cdots \times \mathbb{F}^{m_n}$ (i.e., $\tilde{C}$'s domain) defined as follows: for each $i \in J$, set $\vec{\chi}_i := \vec{\sigma_i}^*$ (where $\vec{\sigma_i}^*$ is the extracted malicious input for party i); for each $i \notin J$, set $\vec{\chi}_i$ to be random in $\mathbb{F}^{m_i}$; output $\vec{\chi} := (\vec{\chi}_i)_{i=1}^n$. The simulator must relay to S a sample from the distribution

$$\mathcal{D} := \left\{ f^{\mathsf{SE}}_{\tilde{C},\mathcal{G}}(\vec{\chi}) \,\middle|\, \vec{\chi} \leftarrow \vec{\mathcal{X}} \right\}_{f^{\mathsf{SS}}_{C,\mathcal{G}}(\vec{\chi}) = f^{\mathsf{SS}}_{C,\mathcal{G}}(\vec{\sigma}^*)} .$$

We now explain how the simulator can efficiently generate a sample from $\mathcal{D}$, despite the fact that the simulator does not know the honest parties' inputs (i.e., the i-th coordinate of $\vec{\sigma}^*$ for $i \notin J$).

By construction of $\tilde{C}$, $f^{\mathsf{SE}}_{\tilde{C},\mathcal{G}}(\vec{\alpha})$ contains $f^{\mathsf{SS}}_{C,\mathcal{G}}(\vec{\alpha})$ for any input $\vec{\alpha}$. However, $f^{\mathsf{SE}}_{\tilde{C},\mathcal{G}}(\vec{\alpha})$ also contains additional outputs; these are the values of wires that carry partial shares of an output of C. Our strategy is to "compute backwards" all the output wires of $\tilde{C}$, starting from its output wires that are also outputs of C (because these values are the ones we know); this strategy leverages the specific structure of the circuit $\tilde{C}$ and does not apply to every circuit in $\mathbf{C}^{\dagger}$. More precisely, we proceed as follows.

Let H be a (potentially empty) subset of $\{1,\dots,n\} \setminus J$ with $|H| = n - |J| - 1$. Initialize $\vec{B}$ to be a vector of $\#\mathsf{inputs}(C)$ components such that the j-th component $\vec{B}[j]$ is a vector of n components where, for $i = 1,\dots,n$, the i-th component $\vec{B}[j,i]$ is chosen as follows: if $i \in J$, then it equals $\sigma^*_{i,j}$; if $i \in H$, then it equals an element drawn uniformly at random from $\mathbb{F}_r$; if i is the single index neither in J nor H then it equals $\perp$. Intuitively, $\vec{B}[j]$ is a multiplicative share of the j-th input of C; we know the shares of malicious parties and, for honest parties, we guess at random for all but one (as we cannot guess at random for all of them, for otherwise we cannot achieve consistency with the output of the trusted party).

Initialize $\vec{E}$ to be a vector of $\#\mathsf{wires}(\tilde{C})$ empty coordinates; the r-th coordinate will be assigned the encoding of the value of the r-th wire in $\tilde{C}$. For each output wire w of C, letting $i := \mathsf{idx}(\mathsf{w})$, do the following:

a) Let g be the gate in $\mathsf{gates}(C)$ that computes $(\alpha^{\mathsf{L}}_0 + \sum_{j=1}^{d^{\mathsf{L}}} \alpha^{\mathsf{L}}_j \mathsf{w}^{\mathsf{L}}_j) \cdot (\alpha^{\mathsf{R}}_0 + \sum_{j=1}^{d^{\mathsf{R}}} \alpha^{\mathsf{R}}_j \mathsf{w}^{\mathsf{R}}_j) \to \mathsf{w}$.

b) Let $\mathcal{R}$ be the $\mathbb{G}$-element that encodes w's value in $f^{\mathsf{SS}}_{C,\mathcal{G}}(\vec{\sigma}^*)$.

c) Set $\vec{E}[\mathsf{idx}(\phi(\mathsf{w}))] := \mathcal{R}$.

d) Use $\vec{B}$ to deduce from $\mathcal{R}$ the encodings of all wires associated to gate g (by referring to the different cases spelled out in the construction of $\tilde{C}$).

4) *Extend the output of* S. Extend $\tilde{\mathsf{tr}}$ with an additional message, $f^{\mathsf{SS}}_{C,\mathcal{G}}(\vec{\sigma}^*)$, and denote the result by tr. This last message reflects the additional round present in Π' (as compared to Π), and its goal is merely to re-format the output of the protocol. Also, since extending $\tilde{\mathsf{tr}}$ to tr does not require additional randomness, we can set $r := \tilde{r}$.

5) *Output.* Output tr (the transcript), $(\vec{\sigma}_i)_{i\in J}$ (the inputs of the corrupted parties), and r (the adversary's randomness).

Appendix B
Proof of Lemma IV.3

We prove Lemma IV.3. Specifically, first we describe the construction of $\Pi^{\mathsf{SE}} = (\Pi, V, S)$ by describing, for every positive integer n and circuit $C \colon \mathbb{F}_r^{m_1} \times \cdots \times \mathbb{F}_r^{m_n} \to \mathbb{F}_r^h$ in $\mathbf{C}^{\dagger}$, the multi-party broadcast protocol $\Pi_{n,C}$, the verifier $V_{n,C}$, and the simulator $S_{n,C}$; afterwards, we explain why the construction of Π^{SE} works.

Below, we use the following cryptographic ingredients: a hiding commitment scheme COMM (see Section III-B); and two NIZKs (see Section III-C), $\mathsf{NIZK}_{\mathscr{R}_{\mathrm{A}}}$ for the NP relation $\mathscr{R}_{\mathrm{A}}$ and $\mathsf{NIZK}_{\mathscr{R}_{\mathrm{B}}}$ for the NP relation $\mathscr{R}_{\mathrm{B}}$. (These two relations are defined in Figure 1.)

Construction of $\Pi_{n,C}$. The n-party broadcast protocol $\Pi_{n,C}$ is a tuple $(S, \Sigma_1, \dots, \Sigma_n)$ that is constructed as follows. The schedule S is

$$S(t) := \begin{cases} \{1,\dots,n\} & \text{if } t = 1 \\ \{i \mid \exists\, \mathsf{w} \in \mathsf{wires}(C) \text{ s.t.} \\ \quad \mathsf{depth}^{\dagger}(\mathsf{w}) = t-1 \\ \quad \text{and } \mathsf{R\text{-}deps}(g_{\mathsf{w}}) = \{i\}\} & \text{if } 1 < t \le \mathsf{depth}^{\dagger}(C) + 1 \\ \{1\} & \text{if } t = \mathsf{depth}^{\dagger}(C) + 2 \\ \emptyset & \text{otherwise} \end{cases} .$$

Next, for $i = 1,\dots,n$, the strategy Σ_i, on input (x_i, t) and with oracle access to the history of messages broadcast so far, works as follows.

- If $t = 1$, do the following. Parse $\mathsf{inputs}(C, i)$ as $\{\mathsf{w}_{i,j}\}_{j=1}^{m_i}$ and x_i as $(\sigma_{i,j})_{j=1}^{m_i}$; each $\sigma_{i,j}$ is an $\mathbb{F}_r$-element and represents the value assigned to wire $\mathsf{w}_{i,j}$. Set $U_i := \big((\mathsf{w}_{i,j}, \sigma_{i,j})\big)_{j=1}^{m_i}$. For $j = 1,\dots,m_i$, sample $(\mathsf{cm}_{i,j}, \mathsf{trap}_{i,j}) \leftarrow \mathsf{COMM.Gen}(\sigma_{i,j})$ and then compute $\pi_{i,j} := \mathsf{NIZK}_{\mathscr{R}_{\mathrm{A}}}.\mathsf{P}(\mathsf{crs}, \mathsf{cm}_{i,j}, (\sigma_{i,j}, \mathsf{trap}_{i,j}))$. Store, for later use, the list U_i and trapdoors $(\mathsf{trap}_{i,1}, \dots, \mathsf{trap}_{i,m_i})$. Output the message $\mathsf{msg}_{t,i} :=$

$(\mathsf{cm}_{i,1}, \pi_{i,1}, \ldots, \mathsf{cm}_{i,m_i}, \pi_{i,m_i})$.

- If $1 < t \leq \mathsf{depth}^{\dagger}(C) + 1$, do the following. Define the set of wires
$$W_{t,i} := \left\{ \mathsf{w} \in \mathsf{wires}(C) \;\middle|\; \begin{array}{c} \mathsf{depth}^{\dagger}(\mathsf{w}) = t-1 \\ \mathsf{R\text{-}deps}(g_{\mathsf{w}}) = \{i\} \end{array} \right\} . \quad (1)$$
If $W_{t,i}$ is empty, halt and do not output any messages (since party i has no gates to process during this round). Otherwise, initialize the message $\mathsf{msg}_{t,i}$ to be an empty list and, for each wire $\mathsf{w} \in W_{t,i}$ taken in topological order, perform the following steps.
  1) Let g be the gate in $\mathsf{gates}(C)$ that computes $(\alpha_0^{\mathsf{L}} + \sum_{j=1}^{d^{\mathsf{L}}} \alpha_j^{\mathsf{L}} \mathsf{w}_j^{\mathsf{L}}) \cdot (\alpha_0^{\mathsf{R}} + \sum_{j=1}^{d^{\mathsf{R}}} \alpha_j^{\mathsf{R}} \mathsf{w}_j^{\mathsf{R}}) \to \mathsf{w}$. Namely, $(\alpha_j^{\mathsf{L}})_{j=0}^{d^{\mathsf{L}}}$ are the left coefficients, $(\alpha_j^{\mathsf{R}})_{j=0}^{d^{\mathsf{R}}}$ the right coefficients, $\{\mathsf{w}_j^{\mathsf{L}}\}_{j=1}^{d^{\mathsf{L}}}$ the left input wires, $\{\mathsf{w}_j^{\mathsf{R}}\}_{j=1}^{d^{\mathsf{R}}}$ the right input wires, and w the (single) output wire.
  2) For $j = 1, \ldots, d^{\mathsf{L}}$, consult the history of messages broadcast so far (or previous iterations of this loop) to obtain a triple $(\mathsf{w}_j, \mathcal{R}_j, \pi_j)$ with $\mathsf{w}_j = \mathsf{w}_j^{\mathsf{L}}$. Allegedly, $\mathcal{R}_j$ encodes w_j's value and π_j is a NIZK proof that attests to this.
  3) Compute the $\mathbb{G}$-element $\mathcal{P} := \alpha_0^{\mathsf{L}} \cdot \mathcal{G} + \sum_{j=1}^{d^{\mathsf{L}}} (\alpha_j^{\mathsf{L}} \cdot \mathcal{R}_j)$; allegedly, $\mathcal{P}$ encodes g's left linear combination.
  4) For $j = 1, \ldots, d^{\mathsf{R}}$, consult U_i to obtain a pair $(\mathsf{w}'_j, \sigma_j)$ with $\mathsf{w}'_j = \mathsf{w}_j^{\mathsf{R}}$; the $\mathbb{F}_r$-element σ_j is w'_j's value.
  5) Compute the $\mathbb{F}_r$-element $\sigma := \alpha_0^{\mathsf{R}} + \sum_{j=1}^{d^{\mathsf{R}}} \alpha_j^{\mathsf{R}} \sigma_j$ and the $\mathbb{G}$-element $\mathcal{R} := \sigma \cdot \mathcal{P}$. Allegedly, $\mathcal{R}$ encodes g's output.
  6) For each $j = 1, \ldots, d^{\mathsf{R}}$:
     – if $\mathsf{w}'_j = \mathsf{w}_{i,j'}$ for some $\mathsf{w}_{i,j'} \in \mathsf{inputs}(C, i)$, set $b_j := 0$, $c_j := \mathsf{cm}_{i,j'}$, and $\mathsf{trap}_j := \mathsf{trap}_{i,j'}$.
     – otherwise, set $b_j := 1$, $c_j := \sigma_j \cdot \mathcal{G}$, and $\mathsf{trap}_j := \perp$.
  7) Set $\vec{b} := (b_j)_{j=1}^{d^{\mathsf{R}}}$, $\vec{\alpha} := (\alpha_j^{\mathsf{R}})_{j=0}^{d^{\mathsf{R}}}$, $\vec{c} := (c_j)_{j=1}^{d^{\mathsf{R}}}$, $\vec{\sigma} := (\sigma_j)_{j=1}^{d^{\mathsf{R}}}$, and $\vec{\mathsf{trap}} := (\mathsf{trap}_j)_{j=1}^{d^{\mathsf{R}}}$.
  8) Construct the instance $\mathbb{x} := (\mathcal{R}, \mathcal{P}, d^{\mathsf{R}}, \vec{\alpha}, \vec{b}, \vec{c})$ and witness $\mathbb{w} := (\vec{\sigma}, \vec{\mathsf{trap}})$. Allegedly, the pair $(\mathbb{x}, \mathbb{w})$ belongs to the NP relation $\mathscr{R}_{\mathrm{B}}$. Compute the NIZK proof $\pi := \mathsf{NIZK}_{\mathscr{R}_{\mathrm{B}}}.\mathsf{P}(\mathsf{crs}', \mathbb{x}, \mathbb{w})$.
  9) Append $(\mathsf{w}, \mathcal{R}, \pi)$ to $\mathsf{msg}_{t,i}$ and (w, σ) to U_i.

  Output the message $\mathsf{msg}_{t,i}$.
- If $t = 2 + \mathsf{depth}^{\dagger}(C)$ and $i = 1$, do the following. Parse $\mathsf{outputs}(C)$ as $\{\mathsf{w}_j^{\mathsf{out}}\}_{j=1}^{h}$. Consult the history of messages broadcast so far to collect the encoding of every output of C, i.e., to collect $\big((\mathsf{w}_j, \mathcal{R}_j, \pi_j)\big)_{j=1}^{h}$ with $\mathsf{w}_j = \mathsf{w}_j^{\mathsf{out}}$. Output the message $\mathsf{msg}_{t,i} := (\mathcal{R}_j)_{j=1}^{h}$.

Construction of $V_{n,C}$. On input a transcript tr, the verifier $V_{n,C}$ first uses $\Pi_{n,C}$'s schedule to parse tr as a sequence of messages $\mathsf{msg}_{t,i}$ where $\mathsf{msg}_{t,i}$ is the t-th message broadcast by party i (or $\perp$ if party i does not act in round t). Here, i ranges from 1 to n, while t ranges from 1 to $2 + \mathsf{depth}^{\dagger}(C)$; if tr cannot be parsed in this way, the verifier rejects. Next, the verifier performs the following checks.

- *Check that parties' inputs are committed.* For $i = 1, \ldots, n$, check that $\mathsf{msg}_{1,i}$ equals a vector of m_i commitments and NIZK proofs, which we denote by $(\mathsf{cm}_{i,1}, \pi_{i,1}, \ldots, \mathsf{cm}_{i,m_i}, \pi_{i,m_i})$. Also check that, for $j = 1, \ldots, m_i$, $\mathsf{NIZK}_{\mathscr{R}_{\mathrm{A}}}.\mathsf{V}(\mathsf{crs}, \mathsf{cm}_{i,j}, \pi_{t,i,j}) = 1$.
- *Check that each gate in C is correctly evaluated.* For $i = 1, \ldots, n$ and $t = 2, \ldots, 1 + \mathsf{depth}^{\dagger}(C)$ do the following. First define the set $W_{t,i}$ as in Equation 1. Then check that $\mathsf{msg}_{t,i}$ equals a list of $|W_{t,i}|$ triples $(\mathsf{w}_{t,i,j}, \mathcal{R}_{t,i,j}, \pi_{t,i,j})$, where each $\mathsf{w}_{t,i,j}$ is a wire of C, each $\mathcal{R}_{t,i,j}$ is an element of $\mathbb{G}$, and each $\pi_{t,i,j}$ is a NIZK proof; also, check that $\{\mathsf{w}_{t,i,j}\}_j = W_{t,i}$. Next, for each j, check that $\mathsf{NIZK}_{\mathscr{R}_{\mathrm{B}}}.\mathsf{V}(\mathsf{crs}', \mathbb{x}_{t,i,j}, \pi_{t,i,j}) = 1$, where $\mathbb{x}_{t,i,j} := (\mathcal{R}_{t,i,j}, \mathcal{P}, d^{\mathsf{R}}, \vec{\alpha}, \vec{b}, \vec{c})$ is an instance for the NP relation $\mathscr{R}_{\mathrm{B}}$ that is constructed as follows. Letting g be the gate of C whose output is $\mathsf{w}_{t,i,j}$, $\mathcal{P}$ equals the linear combination, using g's left coefficients, of the elements of $\mathbb{G}$ that encode the values of g's left inputs (these encodings can be found by consulting suitable parts of the transcript), d^{R} equals the number of right inputs of g, $\vec{\alpha}$ equals the $1 + d^{\mathsf{R}}$ right coefficients of g, $\vec{b}$ is a vector of d^{R} bits in which the k-th bit equals 1 if and only if the k-th right input wire of g is also in $\mathsf{inputs}(C, i)$, and $\vec{c}$ is a vector of d^{R} components in which the k-th component equals $\mathsf{cm}_{i,\ell}$ if the k-th right input wire of g is the ℓ-th wire in $\mathsf{inputs}(C, i)$ and equals $\perp$ if the k-th right input wire of g is not in $\mathsf{inputs}(C, i)$. (The construction of $\mathbb{x}_{t,i,j}$ is analogous to the construction of the NIZK instance in Step 8 above in Σ_i's description.)
- *Check that party 1 collected all the encodings of outputs.* Collect, among the aforementioned triples of the form $(\mathsf{w}, \mathcal{R}, \pi)$, encodings of the values of output wires of C, and check that the vector whose entries equals these encodings matches the message $\mathsf{msg}_{2+\mathsf{depth}^{\dagger}(C),1}$.

Construction of $S_{n,C}$. On input an adversary A and set J of corrupted parties, the simulator $S_{n,C}$ works as follows.

1) *Initialization.* The simulator initializes an empty transcript tr; over the course of running the adversary, the simulator will add to tr both simulated messages (on behalf of honest parties) and messages output by A (on behalf of corrupted parties). The simulator

samples common random strings with an extraction trapdoor for $\mathsf{NIZK}_{\mathscr{R}_{\mathrm{A}}}$ and a simulation trapdoor for $\mathsf{NIZK}_{\mathscr{R}_{\mathrm{B}}}$: $(\mathsf{crs}_{\mathrm{ext}}, \mathsf{trap}_{\mathrm{ext}}) \leftarrow \mathsf{NIZK}_{\mathscr{R}_{\mathrm{A}}}.\mathsf{E}_1$ and $(\mathsf{crs}'_{\mathrm{sim}}, \mathsf{trap}'_{\mathrm{sim}}) \leftarrow \mathsf{NIZK}_{\mathscr{R}_{\mathrm{B}}}.\mathsf{S}_1$. The common random string shown to the adversary is $(\mathsf{crs}_{\mathrm{ext}}, \mathsf{crs}'_{\mathrm{sim}})$. The simulator samples a random string r for the adversary, and then runs the adversary, on inputs $(\vec{\sigma}_i)_{i \in J}$ and with randomness r, until the adversary asks for the first round's messages of the honest parties.

2) *Simulation of the first round.* The adversary expects to receive, for each $i \notin J$, a message $\mathsf{msg}_{1,i}$. The simulator, for each $i \notin J$, does the following. For $j = 1, \ldots, m_i$, sample $(\mathsf{cm}_{i,j}, \mathsf{trap}_{i,j}) \leftarrow \mathsf{COMM.Gen}(\rho_{i,j})$ for a random $\rho_{i,j}$ in $\mathbb{F}_r$ and then compute the NIZK proof $\pi_{i,j} := \mathsf{NIZK}_{\mathscr{R}_{\mathrm{A}}}.\mathsf{P}(\mathsf{crs}_{\mathrm{ext}}, \mathsf{cm}_{i,j}, (\rho_{i,j}, \mathsf{trap}_{i,j}))$; then answer with the message $\mathsf{msg}_{1,i} := (\mathsf{cm}_{i,1}, \pi_{i,1}, \ldots, \mathsf{cm}_{i,m_i}, \pi_{i,m_i})$.
The adversary outputs, for each $i \in J$, a message $\mathsf{msg}_{1,i}$ of his choice. The simulator adds all the third-round messages (the messages that are simulated and those output by the adversary) to the transcript tr.

3) *Invocation of the trusted party.* The simulator, for each $i \in J$, does the following. For $j = 1, \ldots, m_i$, extract $\sigma^*_{i,j}$, the j-th input chosen by the adversary for party i, from the commitment $\mathsf{cm}_{i,j}$ and proof $\pi_{i,j}$ in $\mathsf{msg}_{1,i}$, by computing $\sigma^*_{i,j} := \mathsf{NIZK}_{\mathscr{R}_{\mathrm{A}}}.\mathsf{E}_2(\mathsf{crs}_{\mathrm{ext}}, \mathsf{trap}_{\mathrm{ext}}, \mathsf{cm}_{i,j}, \pi_{i,j})$; then send to the trusted party the vector $\vec{\sigma}_i^{\,*} := (\sigma^*_{i,j})_{j=1}^{m_i}$ as the private input of party i.
At the same time, each honest party $i \in J$ sends to the trusted party his own vector $\sigma_i := (\sigma_{i,j})_{j=1}^{m_i}$. The trusted party, broadcasts the output $f^{\mathsf{SE}}_{C,\mathcal{G}}(\vec{\sigma}^*) := C(\vec{\sigma}^*) \cdot \mathcal{G}$, where $\vec{\sigma}^*$ combines $(\vec{\sigma}_i^{\,*})_{i \in J}$ and $(\vec{\sigma}_i)_{i \notin J}$ in order of i.

4) *Parsing the trusted party's output.* The simulator reorganizes $f^{\mathsf{SE}}_{C,\mathcal{G}}(\vec{\sigma}^*)$ into a data structure that allows for easy lookup of information in the next step: initialize E to be an empty list; then, for each output wire w of C, add to E the pair $(\mathsf{w}, \mathcal{R})$ where $\mathcal{R}$ encodes w's value (and can be found in $f^{\mathsf{SE}}_{C,\mathcal{G}}(\vec{\sigma}^*)$ by definition).

5) *Simulation of the rounds 2 through* $1 + \mathsf{depth}^{\dagger}(C)$. For t ranging from 2 to $1 + \mathsf{depth}^{\dagger}(C)$, the adversary expects to receive, for each $i \notin J$, a message $\mathsf{msg}_{t,i}$. The simulator, for each $i \notin J$, does the following. Define $W_{t,i}$ as in Equation 1. Initialize $\mathsf{msg}_{t,i}$ as an empty list and, for each wire $\mathsf{w} \in W_{t,i}$ taken in topological order, perform the following steps.

   a) Let g be the gate in $\mathsf{gates}(C)$ that computes $(\alpha^{\mathsf{L}}_0 + \sum_{j=1}^{d^{\mathsf{L}}} \alpha^{\mathsf{L}}_j \mathsf{w}^{\mathsf{L}}_j) \cdot (\alpha^{\mathsf{R}}_0 + \sum_{j=1}^{d^{\mathsf{R}}} \alpha^{\mathsf{R}}_j \mathsf{w}^{\mathsf{R}}_j) \to \mathsf{w}$. Namely, $(\alpha^{\mathsf{L}}_j)_{j=0}^{d^{\mathsf{L}}}$ are the left coefficients, $(\alpha^{\mathsf{R}}_j)_{j=0}^{d^{\mathsf{R}}}$ the right coefficients, $\{\mathsf{w}^{\mathsf{L}}_j\}_{j=1}^{d^{\mathsf{L}}}$ the left input wires, $\{\mathsf{w}^{\mathsf{R}}_j\}_{j=1}^{d^{\mathsf{R}}}$ the right input wires, and w the (single) output wire.
   b) For $j = 1, \ldots, d^{\mathsf{L}}$, consult E to obtain the $\mathbb{G}$-element $\mathcal{R}_j$ that encodes $\mathsf{w}^{\mathsf{L}}_j$'s value. Note that $C \in \mathbf{C}^{\dagger}$ implies that each $\mathsf{w}^{\mathsf{L}}_j$ is not in $\mathsf{inputs}(C, i)$, thus its value can be found in E.
   c) For $j = 1, \ldots, d^{\mathsf{R}}$, if $\mathsf{w}^{\mathsf{R}}_j \notin \mathsf{inputs}(C, i)$, consult E to obtain the $\mathbb{G}$-element $\mathcal{S}_j$ that encodes $\mathsf{w}^{\mathsf{R}}_j$'s value.
   d) Consult E to obtain the $\mathbb{G}$-element $\mathcal{R}$ that encodes w's value.
   e) Compute the $\mathbb{G}$-element $\mathcal{P} := \alpha^{\mathsf{L}}_0 \cdot \mathcal{G} + \sum_{j=1}^{d^{\mathsf{L}}} (\alpha^{\mathsf{L}}_j \cdot \mathcal{R}_j)$.
   f) For each $j = 1, \ldots, d^{\mathsf{R}}$: (i) if $\mathsf{w}^{\mathsf{L}}_j = \mathsf{w}_{i,j'}$ for some $\mathsf{w}_{i,j'} \in \mathsf{inputs}(C, i)$, set $b_j := 0$ and $c_j := \mathsf{cm}_{i,j'}$ (where $\mathsf{cm}_{i,j'}$ is in $\mathsf{msg}_{1,i}$). (ii) otherwise, set $b_j := 1$ and $c_j := \mathcal{S}_j$.
   g) Set $\vec{b} := (b_j)_{j=1}^{d^{\mathsf{R}}}$, $\vec{\alpha} := (\alpha^{\mathsf{R}}_j)_{j=0}^{d^{\mathsf{R}}}$, and $\vec{c} := (c_j)_{j=1}^{d^{\mathsf{R}}}$.
   h) Construct the instance $\mathbb{x} := (\mathcal{R}, \mathcal{P}, d^{\mathsf{R}}, \vec{\alpha}, \vec{b}, \vec{c})$ and compute the NIZK proof $\pi := \mathsf{NIZK}_{\mathscr{R}_{\mathrm{B}}}.\mathsf{S}_2(\mathsf{crs}'_{\mathrm{sim}}, \mathsf{trap}'_{\mathrm{sim}}, \mathbb{x})$.
   i) Append $(\mathsf{w}, \mathcal{R}, \pi)$ to $\mathsf{msg}_{t,i}$.

   Answer the adversary with the message $\mathsf{msg}_{t,i}$.
   The adversary outputs, for each $i \in J$, a message $\mathsf{msg}_{t,i}$ of his choice. The simulator adds all the round-t messages (the messages that are simulated and those that are output by the adversary) to the transcript tr.

6) *Simulation of the last round.* If $1 \notin J$, the adversary expects to receive a message $\mathsf{msg}_{2+\mathsf{depth}^{\dagger}(C),1}$; the simulator answers with $\mathsf{msg}_{2+\mathsf{depth}^{\dagger}(C),1} := f^{\mathsf{SE}}_{C,\mathcal{G}}(\vec{\sigma}^*)$. If $1 \in J$, the adversary does not expect any messages and instead outputs a message $\mathsf{msg}_{2+\mathsf{depth}^{\dagger}(C),1}$.
In either case, the simulator adds the the last-round message $\mathsf{msg}_{2+\mathsf{depth}^{\dagger}(C),1}$ to the transcript tr.

7) *Output.* Output tr (the transcript), $(\vec{\sigma}_i)_{i \in J}$ (the inputs of the corrupted parties), and r (the adversary's randomness).

Appendix C
Examples of circuits underlying generators

As discussed in Section I-A, the generator G of essentially all known (preprocessing) zk-SNARK constructions follows the same computation pattern. To generate the public parameters pp for a given NP relation $\mathscr{R}$, G first constructs an $\mathbb{F}_r$-arithmetic circuit $C \colon \mathbb{F}_r^m \to \mathbb{F}_r^h$ (which is somehow related to $\mathscr{R}$), then samples $\vec{\alpha}$ in

$\mathbb{F}_r^m$ at random, and finally outputs $\mathsf{pp} := C(\vec{\alpha}) \cdot \mathcal{G}$ (where $\mathcal{G}$ generates a certain group of order r). Different zk-SNARK constructions differ in (i) which NP relations $\mathscr{R}$ are "natively" supported, and (ii) how the circuit C is obtained from $\mathscr{R}$.

Below, we give two examples of how the generator of a known zk-SNARK construction can be cast in the above paradigm and, moreover, the resulting circuit C lies in the class $\mathbf{C}^\star$. Throughout, we denote by $\mathbb{F}[z]$ the ring of univariate polynomials over $\mathbb{F}$, and by $\mathbb{F}^{\leq d}[z]$ the subring of polynomials of degree $\leq d$.

A. Example for a QAP-based zk-SNARK

We describe how to cast the generator of [21]'s zk-SNARK as computing the encoding of a random evaluation of a circuit C that lies in $\mathbf{C}^\star$. More precisely, we consider [25]'s zk-SNARK, which modifies [21]'s.

Supported NP relations. The zk-SNARK supports relations of the form $\mathscr{R}_D = \{(\vec{x}, \vec{w}) \in \mathbb{F}_r^n \times \mathbb{F}_r^h : D(\vec{x}, \vec{w}) = 0^\ell\}$ where $D \colon \mathbb{F}_r^n \times \mathbb{F}_r^h \to \mathbb{F}_r^\ell$ is an $\mathbb{F}_r$-arithmetic circuit; that is, the zk-SNARK supports arithmetic circuit satisfiability (see Footnote 8).

QAPs. The construction is based on *quadratic arithmetic programs* (QAP) [20]: a QAP of size m and degree d over $\mathbb{F}$ is a tuple $(\vec{A}, \vec{B}, \vec{C}, Z)$, where $\vec{A}, \vec{B}, \vec{C}$ are three vectors, each of $m+1$ polynomials in $\mathbb{F}^{\leq d-1}[z]$, and $Z \in \mathbb{F}[z]$ has degree exactly d. As shown in [20], each relation $\mathscr{R}_D$ can be reduced to a certain relation $\mathscr{R}_{(\vec{A},\vec{B},\vec{C},Z)}$, which captures "QAP satisfiability", by computing $(\vec{A}, \vec{B}, \vec{C}, Z) := \mathsf{GetQAP}(D)$ for a suitable function GetQAP; if D has N_w wires and N_g gates, then the resulting QAP has size $m = N_\mathsf{w}$ and degree $d \approx N_\mathsf{g}$.

The parameter generator. On input an $\mathbb{F}_r$-arithmetic circuit $D \colon \mathbb{F}_r^n \times \mathbb{F}_r^h \to \mathbb{F}_r^\ell$, the generator does:

1) Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \mathsf{GetQAP}(D)$, and denote by m and d the QAP's size and degree; then construct an $\mathbb{F}_r$-arithmetic circuit $C \colon \mathbb{F}_r^8 \to \mathbb{F}_r^{d+7m+n+22}$ such that $C(\tau, \rho_\mathsf{A}, \rho_\mathsf{B}, \alpha_\mathsf{A}, \alpha_\mathsf{B}, \alpha_\mathsf{C}, \beta, \gamma)$ computes the following outputs:

$$\Big(1, \tau, \ldots, \tau^d,$$
$$A_0(\tau)\rho_\mathsf{A}, \ldots, A_m(\tau)\rho_\mathsf{A}, Z(\tau)\rho_\mathsf{A},$$
$$A_0(\tau)\rho_\mathsf{A}\alpha_\mathsf{A}, \ldots, A_m(\tau)\rho_\mathsf{A}\alpha_\mathsf{A}, Z(\tau)\rho_\mathsf{A}\alpha_\mathsf{A},$$
$$B_0(\tau)\rho_\mathsf{B}, \ldots, B_m(\tau)\rho_\mathsf{B}, Z(\tau)\rho_\mathsf{B},$$
$$B_0(\tau)\rho_\mathsf{B}\alpha_\mathsf{B}, \ldots, B_m(\tau)\rho_\mathsf{B}\alpha_\mathsf{B}, Z(\tau)\rho_\mathsf{B}\alpha_\mathsf{B},$$
$$C_0(\tau)\rho_\mathsf{A}\rho_\mathsf{B}, \ldots, C_m(\tau)\rho_\mathsf{A}\rho_\mathsf{B}, Z(\tau)\rho_\mathsf{A}\rho_\mathsf{B},$$
$$C_0(\tau)\rho_\mathsf{A}\rho_\mathsf{B}\alpha_\mathsf{C}, \ldots, C_m(\tau)\rho_\mathsf{A}\rho_\mathsf{B}\alpha_\mathsf{C}, Z(\tau)\rho_\mathsf{A}\rho_\mathsf{B}\alpha_\mathsf{C},$$
$$(A_0(\tau)\rho_\mathsf{A} + B_0(\tau)\rho_\mathsf{B} + C_0(\tau)\rho_\mathsf{A}\rho_\mathsf{B})\beta, \ldots,$$
$$(A_m(\tau)\rho_\mathsf{A} + B_m(\tau)\rho_\mathsf{B} + C_m(\tau)\rho_\mathsf{A}\rho_\mathsf{B})\beta,$$
$$(Z(\tau)\rho_\mathsf{A} + Z(\tau)\rho_\mathsf{B} + Z(\tau)\rho_\mathsf{A}\rho_\mathsf{B})\beta,$$
$$\alpha_\mathsf{A}, \alpha_\mathsf{B}, \alpha_\mathsf{C}, \gamma, \gamma\beta, Z(\tau)\rho_\mathsf{A}\rho_\mathsf{B}, A_0(\tau)\rho_\mathsf{A}, \ldots, A_n(\tau)\rho_\mathsf{A}\Big) .$$

2) Sample $\vec{\alpha}$ in $\mathbb{F}_r^8$ at random.
3) Compute $\mathsf{pp} := C(\vec{\alpha}) \cdot \mathcal{G}$.
4) Output pp.[9]

B. Example for a SSP-based zk-SNARK

We explain how the generator of [31]'s zk-SNARK can be cast as computing the encoding of a random evaluation of a certain circuit C that lies in $\mathbf{C}^\star$.

Supported NP relations. The zk-SNARK supports relations $\mathscr{R}_D = \{(\vec{x}, \vec{w}) \in \{0,1\}^n \times \{0,1\}^h : D(\vec{x}, \vec{w}) = 0^\ell\}$ where $D \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}^\ell$ is a boolean circuit; i.e., the zk-SNARK supports boolean circuit satisfiability (see Footnote 8).

SSPs. The construction is based on *square span programs* (SSP) [31]: a SSP of size m and degree d over $\mathbb{F}$ is a tuple $(\vec{A}, Z)$, where $\vec{A}$ is a vector of $m+1$ polynomials in $\mathbb{F}^{\leq d-1}[z]$ and $Z \in \mathbb{F}[z]$ has degree exactly d. As shown in [31], each relation $\mathscr{R}_D$ can be reduced to a certain relation $\mathscr{R}_{(\vec{A},Z)}$, which captures "SSP satisfiability", by computing $(\vec{A}, Z) := \mathsf{GetSSP}(D)$ for a suitable function GetSSP; if D has N_w wires and N_g gates, then the resulting SSP has size $m = N_\mathsf{w}$ and degree $d \approx N_\mathsf{w} + N_\mathsf{g}$.

The parameter generator. On input a boolean circuit $D \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}^\ell$, the generator does the following.

1) Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \mathsf{GetSSP}(D)$, and denote by m and d the SSP's size and degree; then construct an $\mathbb{F}_r$-arithmetic circuit $C \colon \mathbb{F}_r^3 \to \mathbb{F}_r^{d+2m+n+9}$ such that $C(\tau, \beta, \gamma)$ computes the following outputs:

$$\Big(1, \tau, \ldots, \tau^d,$$
$$A_0(\tau), \ldots, A_m(\tau), Z(\tau),$$
$$A_0(\tau)\beta, \ldots, A_m(\tau)\beta, Z(\tau)\beta,$$
$$\gamma, \gamma\beta, Z(\tau), A_0(\tau), \ldots, A_n(\tau)\Big) .$$

2) Sample $\vec{\alpha}$ in $\mathbb{F}_r^3$ at random.
3) Compute $\mathsf{pp} := C(\vec{\alpha}) \cdot \mathcal{G}$.
4) Output pp.[10]

References

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comp.*, 1989.

[2] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *CRYPTO '92*, 1993.

[3] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," *JACM*, 1991.

[9]The first $d + 7m + 15$ elements in pp form the *proving key* pk, while the remaining $n + 7$ form the *verification key* vk.

[10]The first $d + 2m + 5$ elements in pp form the *proving key* pk, while the remaining $n + 4$ form the *verification key* vk.

[4] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *Journal of Cryptology*, 1994.

[5] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *STOC '88*, 1988.

[6] M. Naor and M. Yung, "Public-key cryptosystems provably secure against chosen ciphertext attacks," in *STOC '90*, 1990.

[7] M. Blum, A. De Santis, S. Micali, and G. Persiano, "Non-interactive zero-knowledge," *SIAM J. Comp.*, 1991.

[8] U. Feige, D. Lapidot, and A. Shamir, "Multiple noninteractive zero knowledge proofs under general assumptions," *SIAM J. Comp.*, 1999.

[9] R. Canetti, R. Pass, and A. Shelat, "Cryptography from sunspots: How to use an imperfect reference string," in *FOCS '07*, 2007.

[10] J. Clark and U. Hengartner, "On the use of financial data as a random beacon," in *EVT/WOTE '10*, 2010.

[11] National Institute of Standards and Technology. (2014) NIST randomness beacon. [Online]. Available: http://www.nist.gov/itl/csd/ct/nist_beacon.cfm

[12] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *STOC '87*, 1987.

[13] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC '88*, 1988.

[14] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *STOC '11*, 2011.

[15] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS '12*, 2012.

[16] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs," in *TCC '13*, 2013.

[17] S. Micali, "Computationally sound proofs," *SIAM J. Comp.*, 2000.

[18] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *ASIACRYPT '10*, 2010.

[19] H. Lipmaa, "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments," in *TCC '12*, 2012.

[20] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT '13*, 2013.

[21] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Oakland '13*, 2013.

[22] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *CRYPTO '13*, 2013.

[23] H. Lipmaa, "Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes," in *ASIACRYPT '13*, 2013.

[24] P. Fauzi, H. Lipmaa, and B. Zhang, "Efficient modular NIZK arguments from shift and product," in *CANS '13*, 2013.

[25] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von Neumann architecture," in *Security '14*, 2014, extended version at http://eprint.iacr.org/2013/879.

[26] H. Lipmaa, "Efficient NIZK arguments via parallel verification of Beneš networks," in *SCN '14*, 2014.

[27] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos, "TRUESET: Faster verifiable set computations," in *Security '14*, 2014.

[28] M. Backes, D. Fiore, and R. M. Reischuk, "Nearly practical and privacy-preserving proofs on authenticated data," ePrint 2014/617, 2014.

[29] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish, "Efficient RAM and control flow in verifiable outsourced computation," ePrint 2014/674, 2014.

[30] Y. Zhang, C. Papamanthou, and J. Katz, "Alitheia: Towards practical verifiable graph processing," in *CCS '14*, 2014.

[31] G. Danezis, C. Fournet, J. Groth, and M. Kohlweiss, "Square span programs with applications to succinct NIZK arguments," in *ASIACRYPT '14*, 2014.

[32] P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *TCC '08*, 2008.

[33] T. Mie, "Polylogarithmic two-round argument systems," *Journal of Mathematical Cryptology*, 2008.

[34] G. Di Crescenzo and H. Lipmaa, "Succinct NP proofs from an extractability assumption," in *CiE '08*, 2008.

[35] I. Damgård, S. Faust, and C. Hazay, "Secure two-party computation with low communication," in *TCC '12*, 2012.

[36] S. Goldwasser, H. Lin, and A. Rubinstein, "Delegation of computation without rejection problem from designated verifier CS-proofs," ePrint 2011/456, 2011.

[37] N. Bitansky and A. Chiesa, "Succinct arguments from multi-prover interactive proofs and their efficiency benefits," in *CRYPTO '12*, 2012.

[38] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for SNARKs and proof-carrying data," in *STOC '13*, 2013.

[39] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," in *CRYPTO '14*, 2014, extended version at http://eprint.iacr.org/2014/595.

[40] N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinstein, and E. Tromer, "The hunting of the SNARK," ePrint 2014/580, 2014.

[41] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Succinct malleable NIZKs and an application to compact shuffles," in *TCC '13*, 2013.

[42] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," in *SOSP '13*, 2013.

[43] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno, "Pinocchio Coin: building Zerocoin from a succinct pairing-based proof system," in *PETShop '13*, 2013.

[44] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *SP '14*, 2014.

[45] M. Fredrikson and B. Livshits, "Zø: An optimizing distributing zero-knowledge compiler," in *Security '14*, 2014.

[46] SCIPR Lab. libsnark: a C++ library for zkSNARK proofs. [Online]. Available: https://github.com/scipr-lab/libsnark

[47] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[48] J. Groth and R. Ostrovsky, "Cryptography in the multi-string model," in *CRYPTO '07*, 2007.

[49] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO '91*, 1992.

[50] J. F. Canny and S. Sorkin, "Practical large-scale distributed key generation," in *EUROCRYPT '04*, 2004.

[51] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, 2007.

[52] A. Kate, Y. Huang, and I. Goldberg, "Distributed key generation in the wild," ePrint 2012/377, 2012.

[53] C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft, "Efficient RSA key generation and threshold Paillier in the two-party setting," in *CT-RSA 2012*, 2012.

[54] J. Katz, A. Kiayias, H.-S. Zhou, and V. Zikas, "Distributing the setup in universally composable multi-party computation," in *PODC '14*, 2014.

[55] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *EUROCRYPT '11*, 2011.

[56] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multi-party computation from somewhat homomorphic encryption," in *CRYPTO '12*, 2012.

[57] R. Lidl and H. Niederreiter, *Finite Fields*, second edition ed. Cambridge University Press, 1997.

[58] C. P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, 1991.

[59] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," in *CRYPTO '87*, 1987.

[60] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO '92*, 1992.

[61] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *SAC'05*, 2006.