Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs

Eli Ben-Sasson[§], Alessandro Chiesa[*], Matthew Green[†], Eran Tromer[¶], Madars Virza[‡]
[*]ETH Zürich, alessandro.chiesa@inf.ethz.ch
[†]Johns Hopkins University, mgreen@cs.jhu.edu
[‡]MIT, madars@mit.edu
[§]Technion, eli@cs.technion.ac.il
[¶]Tel Aviv University, tromer@cs.tau.ac.il

***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 malicious parties to produce 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

In recent years individuals and enterprises have begun to migrate large quantities of internal data to outside providers. This trend raises concerns about the integrity and confidentiality of computations conducted on this data. Consider, for example, the following simple illustrative scenario. A server owns a private database x, and a client wishes to learn $y := F(x)$ for a public function F; a 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], [2]. 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 what is leaked by y).

Non-interactivity. While zero-knowledge proofs can address the above simple scenario, they also apply more broadly, 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

[1]Sometimes a property stronger than soundness is required: *proof of knowledge* [1], [3], 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.

DOI 10.1109/SP.2015.25

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) [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]–[7]. 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 randomness used within G is revealed, then it may be feasible to convince the verifier that false NP statements are true. Compromised soundness can have catastrophic implications, because an attacker may be able to cause significant damage without being detected.

The problem of parameter generation. If no trusted party is available, how should the public parameters pp be generated? Without *some* trustworthy method to generate public parameters, deploying practical systems that rely on NIZKs (e.g., Zerocash [8]) seems very challenging.

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

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

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 problem of parameter generation has garnered recent attention due to the development of new and powerful NIZKs that enable verifying general computation via proofs that are *succinct*, i.e., short and easy to verify [14]. The new proofs are known as *zero-knowledge succinct arguments of knowledge* (zk-SNARKs) [15]–[17], and have already found practical applications, e.g., to building decentralized electronic cash [8]. Most zk-SNARKs require an involved parameter generation, often with complexity proportional to the size of the computation being proved; addressing this parameter generation is the focus of our work. Concretely, we 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 [17]–[31]; in *fully-succinct zk-SNARKs*, that complexity is independent of computation size [14], [16], [32]–[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 [8], [41]–[44].

Public parameters of zk-SNARKs. Despite the aforementioned multitude of constructions, Bitansky et al. [17] showed that essentially all known preprocessing zk-SNARK constructions can be "explained" as the combination of a *linear interactive proof* (LIP) and a cryptographic encoding that only supports 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 (essentially, C is the multi-output circuit that computes the LIP's verifier's message); (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 the 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 n parties are malicious, the protocol's output is pp sampled from the correct distribution and, moreover, the n parties, and any others observing the protocol's execution, learn nothing beyond pp itself. We study this problem, and the special case of generating public parameters for preprocessing zk-SNARKs.

In fact, as the purpose of the multi-party protocol in the aforementioned application is to convince the systems' users, which are bystanders to the protocol execution, we make explicit a *transcript verifier*, which is an algorithm that anyone can use to ensure correct execution of the protocol by examining the broadcast messages.

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 preprocessing zk-SNARKs that includes [21], [25], [31]; we integrated our system with `libsnark` [45], 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 $\mathbf{C}^{\mathsf{S}}$. Roughly, $\mathbf{C}^{\mathsf{S}}$ comprises the $\mathbb{F}$-arithmetic circuits $C\colon \mathbb{F}^m \to \mathbb{F}^h$ for which: (i) the output of each (addition or multiplication) gate is an output of the circuit; (ii) the inputs of each addition gate are outputs of the circuit; (iii) the two inputs of each multiplication gate are, respectively, a circuit output and a circuit input. (See Figure 1a for an example of a circuit in $\mathbf{C}^{\mathsf{S}}$.)

The multi-party protocol is based on standard cryptographic assumptions, and runs atop a synchronous network with an authenticated broadcast channel and a common random string.[3] 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 others.

When n parties participate, our protocol is secure against up to $n-1$ 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.[4] 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 $n \cdot \mathsf{depth}_{\mathsf{S}}(C) + O(1)$ and the number of broadcast messages is $O(n \cdot \mathsf{depth}_{\mathsf{S}}(C))$. Here, $\mathsf{depth}_{\mathsf{S}}(C)$ denotes the S-*depth* of C (introduced later), 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 code implementation is specialized to the case when $\mathbb{G}$ is duplex-pairing because, for this case, several additional optimizations are possible. For this special case, we additionally rely on random oracles so as to benefit from the Fiat–Shamir heuristic [47].

Compared to previous results in secure multi-party computation protocols, our specialized construction scales up to larger number of parties without incurring a high round complexity; see Section II.

(2) Application to zk-SNARKs. Our system can securely sample public parameters of a zk-SNARK, whenever the generator can be cast as sampling the encoding of the random evaluation of a circuit in the class $\mathbf{C}^{\mathsf{S}}$. While the class $\mathbf{C}^{\mathsf{S}}$ 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` [45], and (ii) applied our system to generating public parameters for two specific zk-SNARK constructions: that of [21], [25] (supporting arithmetic relations) and that of [31] (supporting boolean relations). We also extended `libsnark` with an implementation of [31]'s zk-SNARK, augmenting its existing implementation based on [21], [25].

Given an arithmetic circuit D, our code constructs a related circuit C_{PGHR} in $\mathbf{C}^{\mathsf{S}}$, such that the encoding

[3] A broadcast channel can also be thought of as an append-only public logbook and can be implemented in practice, e.g., via Bitcoin's puzzle-based block-chain protocol [46]; authentication can be achieved, e.g., via digital signatures supported by a public-key infrastructure. A common random string can, e.g., be implemented via a public randomness source with high entropy (or even coin-tossing protocols).

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

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 $11N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 38$ and S-depth 3. Similarly, given a boolean circuit D, our code constructs a related circuit C_{DFGK} in $\mathbf{C}^{\mathsf{S}}$ for [31]'s zk-SNARK; if D has N_{w} wires and N_{g} gates, then C_{DFGK} has size $2N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 10$ and S-depth 2.

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

- Our system can securely generate the public parameters for Zerocash [8], a decentralized anonymous payment system extending Bitcoin. Letting D be the circuit that implements the NP relation used in Zerocash: C_{PGHR} has size 138,467,206 and S-depth 3; in our multi-party protocol, the number of rounds is $3n + 3$ and each party works for 14,124 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 8,027,609 and S-depth 6; in our multi-party protocol, the number of rounds is $6n + 6$ and each party works for 4,048 s. (In [39] there are *two* required circuits; here and later we specify, for each complexity measure, the sum of the two costs.)

In both of cases above, the S-depth is extremely small (less than 10), but the standard depth of the same circuit exceeds many hundreds of thousands. The fact that our sampling protocol's round complexity is efficient in S-depth allows for scaling to larger number of parties.

C. Summary of challenges and techniques

We describe at a 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 seek an n-party protocol for sampling $C(\vec{\alpha}) \cdot \mathcal{G}$, for a random $\vec{\alpha}$, that is secure against up to $n - 1$ malicious parties. We may compromise on functionality by restricting C to belong to a circuit class $\mathbf{C}^{\mathsf{S}}$, provided that, in return, we gain improved efficiency (since, ultimately, we want to implement the protocol and use it to generate zk-SNARK public parameters).

The ideal functionality. The first step is to choose the ideal functionality $f_{C,\mathcal{G}}$ to be implemented by the multi-party protocol. A reasonable candidate is the following: 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 field elements, $f_{C,\mathcal{G}}$ outputs the encoding of a random evaluation of C.[5]

Potential approaches. A typical next step is to write a boolean circuit that evaluates $f_{C,\mathcal{G}}$, and then invoke an (off-the-shelf or tailored) multi-party protocol for securely evaluating the circuit. However, the conversion to a boolean is circuit is expensive, because computing $C(\vec{\alpha}) \cdot \mathcal{G}$ involves (i) the evaluation of the $\mathbb{F}_r$-arithmetic circuit C, and (ii) h scalar multiplications over the group $\mathbb{G}$. For example, the number of boolean gates required to compute $C(\vec{\alpha})$ alone is $\geq \log_2 r$ times 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 $\geq \log_2 r$. Similarly, each scalar multiplication over $\mathbb{G}$ expands into a boolean sub-circuit of size $\geq \log_2 q \cdot \log_2 r$. In sum, the conversion incurs a blowup of up to five orders of magnitude in the number of gates to securely evaluate, because $\log_2 q, \log_2 r \geq 250$ (for, e.g., 128 bits of security).

So perhaps we could instead express the computation via an arithmetic circuit, and use a multi-party protocol for arithmetic circuits. However, over what field should the arithmetic circuit be defined? While the circuit C is defined over the field $\mathbb{F}_r$, the group $\mathcal{G}$ may not be; indeed, for the application considered in this paper (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 $\mathbb{G}$ are not. Conversely, if we express the computation as an $\mathbb{F}_q$-arithmetic circuit, while scalar multiplications over $\mathbb{G}$ may be efficient, evaluating C is not. Either way, we again incur the overheads associated to mismatch of field characteristic.

In addition to the above considerations, known multi-party protocols that are secure against malicious majorities either (i) have round complexity that scales linearly with circuit depth, or (ii) rely on heavy cryptographic tools that are unlikely to yield efficient implementations in the near future. The applications that we consider

[5] $f_{C,\mathcal{G}}$ also checks that none of the parties' inputs contains a zero. Forbidding zeros biases the output distribution, but only negligibly, since r is chosen large enough for discrete log to be hard in $\mathbb{Z}_r^*$.

involve circuit depths exceeding hundreds of thousands, so that such works do not seem applicable (see Section II for discussion and citations).

Our approach. Our approach avoids the overheads due to mismatch in field characteristic, and also has low round complexity.

We observe that, for particular zk-SNARK constructions (including [21], [25], [31]) the circuit C can (as discussed in Section I-C2), be written to have a special form so as to lie in the circuit class $\mathbf{C}^{\mathsf{S}}$. We restrict our attention to implementing $f_{C,\mathcal{G}}$ for $C \in \mathbf{C}^{\mathsf{S}}$.

For such circuits, we design a protocol where parties jointly homomorphically evaluate the circuit C (avoiding, in particular, first computing $\vec{\beta} := C(\vec{\alpha})$ and then $\vec{\beta} \cdot \mathcal{G}$). First, all parties first 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 input share. Additions are done locally (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 scheduling of when each party contributes his own share, we can reduce the number of rounds to only n times C's S-depth, where S-depth is a much milder notion of depth (defined later). In the zk-SNARK application that we consider, $\mathsf{depth}(C)$ grows with $\mathsf{size}(C)$ while $\mathsf{depth}_{\mathsf{S}}(C)$ is a small constant.

We realize the above approach by splitting the construction in two steps. First, we reduce the problem of sampling the encoding of a random evaluation of C to the problem of jointly evaluating a related circuit $\tilde{C}$. Second, we build a multi-party protocol for securely evaluating $\tilde{C}$. These two steps simplify providing a formal proof of security, as well as building a prototype implementation of the protocol. Section I-D summarizes our construction.

Our implementation is specialized to when $\mathbb{G}$ is a duplex-pairing group, in which case the NIZKs used by parties can be implemented very efficiently via Schnorr proofs and the Fiat–Shamir heuristic [47].

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 a procedure for transforming the NP relation (represented as an instance D of arithmetic or boolean satisfiability) given as input to generator, into a corresponding circuit $C \in \mathbf{C}^{\mathsf{S}}$ such that $C(\vec{\alpha}) \cdot \mathcal{G}$ for a random $\vec{\alpha}$ equals the distribution of public parameters output by the generator.

Constructing such a circuit C, subject to the restrictions of $\mathbf{C}^{\mathsf{S}}$ (needed for applying our sampling protocol), is not straightforward for either of the aforementioned zk-SNARKs. One issue that arises, in both cases, is how to construct a sub-circuit that, given an input $\tau \in \mathbb{F}_r$, evaluates all Lagrange interpolating polynomials at τ; indeed, the standard linear-size circuit for this operation involves division gates, which our protocol does not handle (and are thus not included in $\mathbf{C}^{\mathsf{S}}$). Instead of relying on the standard circuit, we rely on a suitable FFT-like sub-circuit that avoids the division gates.

D. Construction summary

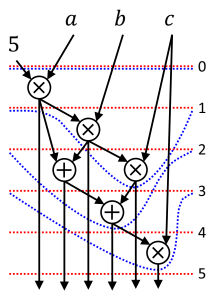
We summarize our construction of an n-party protocol for sampling $\mathsf{pp} := C(\vec{\alpha}) \cdot \mathcal{G}$, for a random $\vec{\alpha}$, that is secure against up to $n-1$ malicious parties. Recall that we focus on circuits in the class $\mathbf{C}^{\mathsf{S}}$, which consists of circuits $C \colon \mathbb{F}^m \to \mathbb{F}^h$ for which: (i) the output of each (addition or multiplication) gate is an output of the circuit; (ii) the inputs of each addition gate are outputs of the circuit; (iii) the two inputs of each multiplication gate are, respectively, a circuit output and a circuit input. See Figure 1a for an example of a circuit in $\mathbf{C}^{\mathsf{S}}$.

We first introduce some ideas for the artificial special case of a single party executing the protocol; we then explain how these ideas can be extend to multiple parties.

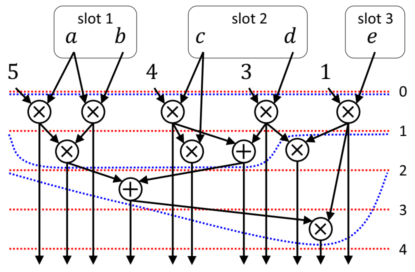
A special case. Suppose that a single party wishes to generate pp in a verifiable way: the party outputs a *transcript* tr, from which pp can be deduced, such that anyone can establish validity of the transcript. Informally, we seek a *verifier* V and *simulator* S that satisfy: (1) *syntactical correctness*: if $V(\mathsf{tr}) = 1$, there is $\vec{\alpha} \in \mathbb{F}^m$ such that $\mathsf{pp} = C(\vec{\alpha}) \cdot \mathcal{G}$; and (2) *zero knowledge*: tr reveals no information beyond pp in the sense that $S(\mathsf{pp})$ is indistinguishable from tr. (At this stage we do not yet ensure that $\vec{\alpha}$ is uniformly drawn and unknown.)

The straightforward approach to achieve the above is to set $\mathsf{tr} := (\mathsf{pp}, \pi)$ where π is a NIZK proof (in the common *random* string model) for the NP statement "there exists $\vec{\alpha}$ such that $\mathsf{pp} = C(\vec{\alpha}) \cdot \mathcal{G}$", and then to let V be the NIZK verifier and S be the NIZK simulator.

We observe that, since C is in $\mathbf{C}^{\mathsf{S}}$, the NP statement above can be "factored" into a collection of sub-statements so that the proof π can be constructed as the concatenation of a NIZK sub-proof π_g for each gate g in C. Essentially, for each gate g taken in topological order: if g is a multiplication gate, then we know that the encoding $\mathcal{P}$ of one of the two inputs has already been computed (and proved correct) so that, if $\gamma \in \mathbb{F}$ denotes the other input and $\mathcal{R}$ the encoded output, the party can generate a NIZK proof that $\mathcal{R} = \gamma \cdot \mathcal{P}$; if instead g is an



(a) Example of a circuit in $\mathbf{C}^{\mathsf{S}}$.



(b) Example of a circuit in $\mathbf{C}^{\mathsf{E}}$.

Fig. 1: Examples of a circuit in $\mathbf{C}^{\mathsf{S}}$ and one in $\mathbf{C}^{\mathsf{E}}$; in the latter case, the inputs of the circuit are partitioned into slots. The red contour lines denote (traditional) circuit depth, while blue contour lines denote S-depth and E-depth.

addition gate, the group structure of $\mathbb{G}$ enables anyone to evaluate g so that no NIZK proof is needed.

Extending to multiple parties. Now suppose that there are $n > 1$ participating parties. Denote by tr the transcript of all broadcast messages. We seek a verifier V and simulator S that satisfy variants of the above properties: (1) *distributional correctness*: if at least one party is honest and $V(\mathsf{tr}) = 1$, pp equals $C(\vec{\alpha}) \cdot \mathcal{G}$ for a random $\vec{\alpha}$; and (2) *zero knowledge*: if at least one party is honest and $V(\mathsf{tr}) = 1$, tr reveals no information beyond pp in the sense that $S(\mathsf{pp})$ is indistinguishable from tr.

Ideally, we would still set $\mathsf{tr} := (\mathsf{pp}, \pi)$ where π is a NIZK proof for the NP statement "there exists $\vec{\alpha}$ such that $\mathsf{pp} = C(\vec{\alpha}) \cdot \mathcal{G}$". However, now there is no single party that knows the witness $\vec{\alpha}$. Each party holds a multiplicative share of every coordinate of $\vec{\alpha}$: party i holds $\vec{\sigma}_i = (\sigma_{i,1}, \ldots, \sigma_{i,m}) \in \mathbb{F}_r^m$ and α_j equals $\prod_{i=1}^{n} \sigma_{i,j}$.

Nevertheless, we show that it is still possible to factor the NP statement into a collection of sub-statements, each one involving a contribution of one share, that, when carefully scheduled, allow the n parties to jointly assemble π by producing suitable sub-proofs. More generally, our construction has two steps. First, we transform the circuit C in $\mathbf{C}^{\mathsf{S}}$ into a new circuit $\tilde{C}$ in a new circuit class $\mathbf{C}^{\mathsf{E}}$; second, we construct a multi-party protocol for securely evaluating any circuit in $\mathbf{C}^{\mathsf{E}}$. More details follow.

The circuit class $\mathbf{C}^{\mathsf{E}}$ differs from $\mathbf{C}^{\mathsf{S}}$ in two ways. First, the inputs of a circuit $\tilde{C} \in \mathbf{C}^{\mathsf{E}}$ are partitioned into *slots*; we write $\tilde{C} \colon \mathbb{F}^{m_1} \times \cdots \times \mathbb{F}^{m_n} \to \mathbb{F}^h$ to express that the first m_1 inputs are in the first slot, the next m_2 in the second, and so on; the integers $m_1, \ldots, m_n$ are part of $\tilde{C}$'s description. Second, the restriction on the possible inputs of multiplication gates is relaxed to account for input slots. $\mathbf{C}^{\mathsf{E}}$ poses the following restrictions on $\tilde{C}$ (with difference from $\mathbf{C}^{\mathsf{S}}$ emphasized): (i) the output of each (addition or multiplication) gate is an output of the circuit; (ii) the inputs of each addition gate are outputs of the circuit; (iii) the two inputs of each multiplication gate are, respectively, a circuit output and either a circuit input, *or a circuit output computable from inputs from a single slot*. Figure 1b is an example of a circuit in $\mathbf{C}^{\mathsf{E}}$.

The transformation from $C \in \mathbf{C}^{\mathsf{S}}$ to $\tilde{C} \in \mathbf{C}^{\mathsf{E}}$ is as follows. The m inputs of C are multiplicatively shared among n parties to obtain $n \cdot m$ inputs for $\tilde{C}$; the slot i of $\tilde{C}$ contains the m shares of party i. Each multiplication gate in C is mapped to $O(n)$ multiplication gates in $\tilde{C}$ tasked with assembling all the relevant shares; each addition gate in C is mapped to a corresponding addition gate in $\tilde{C}$. A crucial feature of the transformation is depth efficiency (see below).

The multi-party protocol for circuits in $\mathbf{C}^{\mathsf{E}}$ is a generalization of the one that we described above for a single party. Essentially, the class $\mathbf{C}^{\mathsf{E}}$ ensures that at each multiplication gate there is one party that knows the "local" witness for producing a NIZK proof of correct evaluation of the gate. Thus, the protocol proceeds in rounds, and at each round every party proves correct evaluation of any gate ready to be processed (and so on until no more gates need to be processed).

Depth matters. The round complexity of securely evaluating $\tilde{C} \in \mathbf{C}^{\mathsf{E}}$ is $\mathsf{depth}_{\mathsf{E}}(\tilde{C}) + O(1)$, where $\mathsf{depth}_{\mathsf{E}}(\tilde{C})$ is the E-*depth* and (roughly) corresponds to the maximum number of *gate-ownership* alternations along any input-to-output path; ownership refers to which party provides the input share to a gate. (See Figure 1b for a comparison of depth and E-depth for an example in $\mathbf{C}^{\mathsf{E}}$.) Intuitively, while going down a path in the circuit, every change in gate ownership means that a party needs to wait on another one to process the previous gate, thereby costing an extra round.

Therefore, it is crucial that the transformation from C to $\tilde{C}$ is efficient in terms of E-depth of $\tilde{C}$. By carefully combining the sub-circuits in $\tilde{C}$, we ensure that $\mathsf{depth}_{\mathsf{E}}(\tilde{C}) = n \cdot \mathsf{depth}_{\mathsf{S}}(C)$, where $\mathsf{depth}_{\mathsf{S}}(C)$ is the S-*depth* of C and denotes the maximum number of alternations between addition and multiplication gates along any input-to-output path. (Figure 1a compares depth and S-depth for an example in $\mathbf{C}^{\mathsf{S}}$.)

II. Prior work

Secure generation of parameters. Generating public parameters for NIZKs has been studied before, particularly in the setting where parameters merely consist of a random string. For example, [9], [10], [48] 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]–[51].

Secure multi-party computation. The area of secure multi-party computation has seen rapid recent progress, both in terms of theoretical results and concrete implementations. Yet, the existing generic implementations do not support, or inefficiently support, the setting that we consider: many parties, dishonest majority, and evaluation of a circuit with large (standard) circuit depth.

For example, many implementations consider the case of two parties [52], where they achieve outstanding efficiency [53], [54], and can process billions of boolean gates while spending only tens of CPU cycles on each. Most of the approaches in this setting are based on Yao's seminal work on garbled circuits [55], [56].

Some implementations consider the case of arbitrary number of parties, but they suffer from other limitations. For example, [57] consider adversaries that are honest but curious. Other protocols [13], [58] consider malicious adversaries but require an honest majority. There are known constant-round MPC protocols for a fully-malicious, dishonest majority [59], [60], but these require expensive ZK proofs and have not been implemented.

When requiring security against dishonest majorities (with at least one honest party), implementations have a round complexity that depends linearly on the depth of the circuit being computed [61]–[65]. The applications that we consider in this paper involve circuit depths that exceed hundreds of thousands, resulting in large round complexities; such round complexities are at best very expensive when considering network latencies on the Internet and at worst prohibitive if one of the participating parties uses an air gap as a precaution. While theoretical results do achieve sublinear round complexity [66], [67], they rely on "heavy artillery" such as fully-homomorphic encryption and program obfuscation, unlikely to yield efficient implementations in the near future.

III. Definitions

A. Basic notation

We denote by λ the security parameter; $f = O_{\lambda}(g)$ means that there exists $c > 0$ such that $f = O(\lambda^{c} g)$. The power set of a set S is denoted 2^{S}. Vectors are denoted by arrow-equipped letters (e.g., $\vec{a}$); their entries carry an index but not the arrow (e.g., a_1, a_2). Concatenation of vectors (and scalars) is denoted by the operator $\circ$.

Implicit inputs. To simplify notation, the input 1^{λ} is implicit to all cryptographic algorithms; similarly, we do not make explicit adversaries' auxiliary inputs.

Distributions. We write $\{y \,|\, x_1 \leftarrow D_1\,;\, x_2 \leftarrow D_2\,;\, \dots\}_E$ 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 $y := y(x_1, x_2, \dots)$. 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 the 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 consider only groups that are cyclic and have a prime order r. Group elements are denoted with calligraphic letters (such as $\mathcal{P}, \mathcal{Q}$). 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. Hence, $\mathcal{P} + \mathcal{Q}$ denotes addition of the two elements $\mathcal{P}$ and $\mathcal{Q}$; $a \cdot \mathcal{P}$ denotes scalar multiplication of $\mathcal{P}$ by the scalar $a \in \mathbb{Z}$; and $\mathcal{O} := 0 \cdot \mathcal{P}$ denotes the identity element. Since $r \cdot \mathcal{P} = \mathcal{O}$, we can equivalently think of a scalar a as belonging to the field of size r. Given a vector $\vec{a} = (a_1, \dots, a_n)$, we use $\vec{a} \cdot \mathcal{P}$ as a shorthand for the vector $(a_1 \cdot \mathcal{P}, \dots, a_n \cdot \mathcal{P})$.

Fields. We denote by $\mathbb{F}$ a field, and by $\mathbb{F}_n$ the field of size n; we consider only fields of prime order. Field elements are denoted with Greek letters (such as α, β, γ).

B. Commitments

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

- $\mathsf{COMM.Gen}(x) \to (\mathsf{cm}, \mathsf{cr})$*:* On input data x, the *commitment generator* $\mathsf{COMM.Gen}$ probabilistically samples a commitment cm of x and corresponding commitment randomness cr.
- $\mathsf{COMM.Ver}(x, \mathsf{cm}, \mathsf{cr}) \to b$*:* On input data x, commitment cm, and commitment randomness cr, the *commitment verifier* $\mathsf{COMM.Ver}$ outputs $b = 1$ if

cm is a valid commitment of x with respect to the randomness cr.

The scheme COMM satisfies the standard completeness, (computational) binding, and (statistical) hiding properties. We do not assume that cm hides $|x|$.

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 $O_\lambda(1)$ bits (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 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 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 $\exists\, \mathbb{w}$ s.t. $(\mathbb{x}, \mathbb{w}) \in \mathscr{R}$), the *simulator* $\mathsf{NIZK}_{\mathscr{R}}.\mathsf{S}_2$ outputs π that is indistinguishable from an "honest" proof.

$\mathsf{NIZK}_{\mathscr{R}}$ satisfies the standard completeness, (computational, adaptive) proof-of-knowledge, and (statistical, adaptive, multi-theorem) zero-knowledge properties.

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 $\mathsf{size}(C) := \#\mathsf{inputs}(C) + \#\mathsf{gates}(C)$.

Gate types. A gate g of C is an *addition gate* g_{add}, of the form $\alpha_0 + \sum_{j=1}^{d} \alpha_j \mathsf{w}_j$, or a (2-input) *multiplication gate* g_{mul}, of the form $\alpha \mathsf{w}^{\mathsf{L}} \mathsf{w}^{\mathsf{R}}$. For addition gates, $\mathsf{inputs}(g_{\mathsf{add}}) := \{\mathsf{w}_1, \dots, \mathsf{w}_d\}$ are the input wires and $\mathsf{coeffs}(g_{\mathsf{add}}) := (\alpha_0, \dots, \alpha_d)$ are the coefficients. For multiplication gates, $\mathsf{L\text{-}input}(g_{\mathsf{mul}}) := \mathsf{w}^{\mathsf{L}}$ is the left input, $\mathsf{R\text{-}input}(g_{\mathsf{mul}}) := \mathsf{w}^{\mathsf{R}}$ is the right input, and $\mathsf{coeffs}(g_{\mathsf{mul}}) := (\alpha)$ is the coefficient. For both gate types, $\mathsf{output}(g) := \mathsf{w}$ is the output wire; also, g_{w} is the gate for which $\mathsf{w} = \mathsf{output}(g_{\mathsf{w}})$. We define $\mathsf{type}(g)$ to be add for addition gates, and mul for multiplication gates; constant gates (implicit in figures) are a special case of addition gates.

Further notions for circuits with partitioned domains. We also consider $\mathbb{F}$-arithmetic circuits C for which the m inputs of the circuits are partitioned into n disjoint *slots*; 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 belong to the first slot, the next m_2 to the second, and so on; the integers $m_1, \dots, m_n$ are then also part of C's description. For $i = 1, \dots, n$: we denote by $\mathsf{inputs}(C, i)$ the input wires that belong to the i-th slot, 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 $\mathsf{size}(C, i) := \#\mathsf{inputs}(C, i) + \#\mathsf{gates}(C, i)$. For every $\mathsf{w} \in \mathsf{inputs}(C)$, $\mathsf{input\text{-}slot}(C, \mathsf{w})$ is w's slot number, i.e., the index i such that $\mathsf{w} \in \mathsf{inputs}(C, i)$.

Finally, to assist in stating the definition of E-depth (see below), we introduce the *dependency set* $\mathsf{ds}(\mathsf{w})$ of a wire w; roughly, it denotes the subset of $\{1, \dots, n\}$ denoting which slots individually carry enough information (in terms of inputs) to compute the value of w. The formal definition of $\mathsf{ds}(\mathsf{w})$ is quite technical, and is Figure 2.

Two classes of circuits. We consider the following two circuit classes $\mathbf{C}^{\mathsf{S}}$ and $\mathbf{C}^{\mathsf{E}}$.

- $\mathbf{C}^{\mathsf{S}}$ is 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) if $\mathsf{type}(g) = \mathsf{add}$, then $\mathsf{inputs}(g) \cap \mathsf{inputs}(C) = \emptyset$; and (iii) if $\mathsf{type}(g) = \mathsf{mul}$, then $\mathsf{L\text{-}input}(g) \notin \mathsf{inputs}(C)$ and $\mathsf{R\text{-}input}(g) \in \mathsf{inputs}(C)$.
- $\mathbf{C}^{\mathsf{E}}$ is 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) if $\mathsf{type}(g) = \mathsf{add}$, then $\mathsf{inputs}(g) \cap \mathsf{inputs}(C) = \emptyset$; and (iii) if $\mathsf{type}(g) = \mathsf{mul}$, then $\mathsf{L\text{-}input}(g) \notin \mathsf{inputs}(C)$ and, for

every $\mathsf{w}, \mathsf{w}' \in \mathsf{inputs}(C)$, if $\mathsf{R\text{-}input}(g)$ depends on w and w' then $\mathsf{input\text{-}slot}(C, \mathsf{w}) = \mathsf{input\text{-}slot}(C, \mathsf{w}')$.

Notions of depth. For circuits in $\mathbf{C}^{\mathsf{S}}$ and $\mathbf{C}^{\mathsf{E}}$, we use alternative notions of depth, called S-*depth* and E-*depth*; both S-depth and E-depth are bounded from above by (traditional) circuit depth, but are sometimes much less.

- The S-depth of C in $\mathbf{C}^{\mathsf{S}}$ is $\mathsf{depth}_{\mathsf{S}}(C) := \max_{\mathsf{w} \in \mathsf{outputs}(C)} \mathsf{depth}_{\mathsf{S}}(\mathsf{w})$ and $\mathsf{depth}_{\mathsf{S}}(\mathsf{w})$ is defined in Figure 2; there, $b_{\mathsf{w}} \in \{0,1\}$ equals 1 if and only if either $|\mathsf{inputs}(g_{\mathsf{w}})| \geq 2$ or $|\mathsf{inputs}(g_{\mathsf{w}})| = 1 \wedge \mathsf{coeffs}(g)[0] \neq 0$
- The E-depth of C in $\mathbf{C}^{\mathsf{E}}$ is $\mathsf{depth}_{\mathsf{E}}(C) := \max_{\mathsf{w} \in \mathsf{outputs}(C)} \mathsf{depth}_{\mathsf{E}}(\mathsf{w})$ and $\mathsf{depth}_{\mathsf{E}}(\mathsf{w})$ is defined in Figure 2; there, $b^{\mathsf{add}}_{\mathsf{w}} \in \{0,1\}$ equals 1 if and only if $\mathsf{ds}(\mathsf{L\text{-}input}(g_{\mathsf{w}})) \cap \mathsf{ds}(\mathsf{R\text{-}input}(g_{\mathsf{w}})) = \emptyset$ and $\mathsf{depth}_{\mathsf{E}}(\mathsf{L\text{-}input}(g_{\mathsf{w}})) \leq \mathsf{depth}_{\mathsf{E}}(\mathsf{R\text{-}input}(g_{\mathsf{w}}))$, and $b^{\mathsf{mul}}_{\mathsf{w}} \in \{0,1\}$ equals 1 if and only if $\bigcap_{\mathsf{w}' \in \mathsf{inputs}(g_{\mathsf{w}})} \mathsf{ds}(\mathsf{w}') = \emptyset$.

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. $e(\mathcal{G}_1, \mathcal{G}_2) \neq 1$

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 $\{(\alpha \cdot \mathcal{G}_1, \alpha \cdot \mathcal{G}_2) \,|\, \alpha \in \mathbb{F}_r\} \subseteq \mathbb{G}_1 \times \mathbb{G}_2$.

F. Secure multi-party computation

We specialize definitions of secure multi-party computation [12], [13] to our setting, by considering parties' inputs that are field elements rather than bit strings, by considering families of functionalities rather than a single functionality, and making explicit the notion of a (transcript) verifier. These definitions provide background and notation for this paper (and closely follow the treatment in [68]). We assume familiarity with simulation-based security definitions; for more, see [68].

1) 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; 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) $t := t+1$.

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 $[\![\Pi, \vec{x}]\!]_{A,J}$.

2) Ideal functionalities

While Section III-F1 describes the real-world execution of a protocol Π on an input $\vec{x}$, here we describe the

$$\mathsf{ds}(\mathsf{w}) := \begin{cases} \{\mathsf{input\text{-}slot}(C, \mathsf{w})\} & \text{if } \mathsf{w} \in \mathsf{inputs}(C) \\ \{\mathsf{input\text{-}slot}(C, \mathsf{R\text{-}input}(g_{\mathsf{w}}))\} & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{mul} \\ \{1, \ldots, n\} & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{add}, |\mathsf{inputs}(g_{\mathsf{w}})| = 0 \\ \bigcap_{\mathsf{w}' \in \mathsf{inputs}(g_{\mathsf{w}})} \mathsf{ds}(\mathsf{w}') & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{add}, \bigcap_{\mathsf{w}' \in \mathsf{inputs}(g_{\mathsf{w}})} \mathsf{ds}(\mathsf{w}') \neq \emptyset \\ \{1, \ldots, n\} & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{add}, \bigcap_{\mathsf{w}' \in \mathsf{inputs}(g_{\mathsf{w}})} \mathsf{ds}(\mathsf{w}') = \emptyset \end{cases}$$

$$\mathsf{depth}_{\mathsf{S}}(\mathsf{w}) := \begin{cases} 1 & \text{if } \mathsf{w} \in \mathsf{inputs}(C) \\ b_{\mathsf{w}} + \max \ \{\mathsf{depth}_{\mathsf{S}}(\mathsf{w}')\}_{\mathsf{w}' \in \mathsf{inputs}(g_{\mathsf{w}})} & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{add} \\ \max\left(\mathsf{depth}_{\mathsf{S}}(\mathsf{L\text{-}input}(g_{\mathsf{w}})), \mathsf{depth}_{\mathsf{S}}(\mathsf{R\text{-}input}(g_{\mathsf{w}}))\right) & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{mul} \end{cases}$$

$$\mathsf{depth}_{\mathsf{E}}(\mathsf{w}) := \begin{cases} 1 & \text{if } \mathsf{w} \in \mathsf{inputs}(C) \\ 0 & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{add}, |\mathsf{inputs}(g_{\mathsf{w}})| = 0 \\ b_{\mathsf{w}}^{\mathsf{mul}} + \max_{\mathsf{w}' \in \mathsf{inputs}(g_{\mathsf{w}})} \mathsf{depth}_{\mathsf{E}}(\mathsf{w}') & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{add}, |\mathsf{inputs}(g_{\mathsf{w}})| > 0 \\ \max\left(\mathsf{depth}_{\mathsf{E}}(\mathsf{L\text{-}input}(g_{\mathsf{w}})), \mathsf{depth}_{\mathsf{E}}(\mathsf{R\text{-}input}(g_{\mathsf{w}})) + b_{\mathsf{w}}^{\mathsf{add}}\right) & \text{if } \mathsf{w} \notin \mathsf{inputs}(C), \mathsf{type}(g_{\mathsf{w}}) = \mathsf{mul} \end{cases}$$

Fig. 2: Definitions of dependency set, S-depth, and E-depth; see Section III-D.

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 $\{1, \ldots, n\}$. 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.

3) *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 $\Pi^{\mathsf{S}} = (\Pi, V, S)$, 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 $\{1, \ldots, n\}$ (the *corrupted parties*) with $|J| < n$, these two distributions are negligibly close:

$$\left\{ \mathsf{REAL}_{\Pi,A,J}(\vec{\sigma}) \;\middle|\; \begin{array}{c} \vec{\sigma}_1 \leftarrow \mathbb{F}_r^m \\ \vdots \\ \vec{\sigma}_n \leftarrow \mathbb{F}_r^m \end{array} \right\}_{V=1}$$

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

Above, $\vec{\sigma}$ denotes $(\vec{\sigma}_1, \ldots, \vec{\sigma}_n)$; $V = 1$ denotes conditioning on the event that V, on input the transcript of $[\![\Pi, \vec{x}]\!]_{A,J}$, outputs 1; and $f^{\mathsf{S}}_{C,\mathcal{G}}$ denotes the deterministic function such that $f^{\mathsf{S}}_{C,\mathcal{G}}(\sigma) := C((\prod_{i=1}^{n} \sigma_{i,1}, \ldots, \prod_{i=1}^{n} \sigma_{i,m})) \cdot \mathcal{G}$.

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 $\Pi^{\mathsf{S}} = (\Pi, V, S)$ 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 $\mathbb{G}$.

4) *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 $\Pi^{\mathsf{E}} = (\Pi, V, S)$, 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 $\{1, \ldots, n\}$ with $|J| < n$, and input $\vec{\sigma} = (\vec{\sigma}_1, \ldots, \vec{\sigma}_n)$ in $\mathbb{F}_r^{m_1} \times \cdots \times \mathbb{F}_r^{m_n}$,

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

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

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 $\Pi^{\mathsf{E}} = (\Pi, V, S)$ 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 $\mathbf{C}^{\mathsf{S}}$. We use two cryptographic ingredients: commitment schemes (see Section III-B) and NIZKs (see Section III-C); both rely on a common *random* string, available in our setting (see Section III-F1).

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

- $\mathsf{ROUND}(\Pi_{n,C}) = n \cdot \mathsf{depth}_{\mathsf{S}}(C) + 3$.
- *for $i = 1, \ldots, n$, $\mathsf{TIME}(\Pi_{n,C}, i) = O_\lambda(\mathsf{size}(C))$.*
- *$V_{n,C}$ runs in time $O_\lambda(n \cdot \mathsf{size}(C))$.*
- *$S_{n,C}$ runs in time $O_\lambda(n \cdot \mathsf{size}(C))$.*

Our implementation and evaluation target the case when $\mathbb{G}$ is a duplex-pairing group. This special case allows for additional optimizations (when further relying on random oracles), as discussed in Section V.

Proof strategy. We construct the protocol of Theorem IV.1 in two steps. The first step (Lemma IV.2) 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 for which the following holds.*

- *For every positive integer n and circuit C in $\mathbf{C}^{\mathsf{S}}$: (i) $\tilde{C} := T_1(n, C)$ is a circuit in $\mathbf{C}^{\mathsf{E}}$; (ii) for every secure evaluation broadcast protocol Π^{E} with n parties for $\tilde{C}$ over $\mathbb{G}$, $\Pi^{\mathsf{S}} := T_2(\Pi^{\mathsf{E}})$ is a secure sampling broadcast protocol with n parties for C over $\mathbb{G}$.*
- *T_1 builds a new circuit $\tilde{C}$ is not much larger than C:*
  - $\mathsf{depth}_{\mathsf{E}}(\tilde{C}) = n \cdot \mathsf{depth}_{\mathsf{S}}(C)$*;*
  - $\mathsf{size}(\tilde{C}) = O(n \cdot \mathsf{size}(C))$*; and*
  - $\mathsf{size}(\tilde{C}, i) = O(\mathsf{size}(C))$ *for $i = 1, \ldots, n$.*
- *T_2 increases the protocol's round complexity by 1, and preserves all time complexities up to $O_\lambda(1)$ factors.*

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

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

A. Sketch of the sampling-to-evaluation reduction

We sketch the proof of Lemma IV.2. At a 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 $\mathbf{C}^{\mathsf{S}}$, outputs a circuit $\tilde{C} \in \mathbf{C}^{\mathsf{E}}$ 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^{E} for $\tilde{C}$, outputs a secure sampling protocol Π^{S} for C by: (i) generating random shares for all inputs, to ensure uniform sampling; (ii) extending the protocol by one last 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).

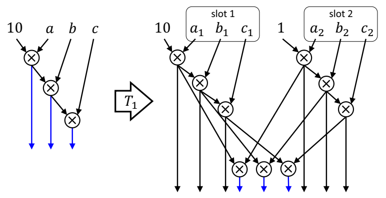
Most of the effort goes into constructing $\tilde{C}$ and the simulator of Π^{S}. We thus briefly discuss these two.

The circuit $\tilde{C}$. The circuit $\tilde{C}$ must compute C's output from n multiplicative shares of C's input (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 shares 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 $\mathbf{C}^{\mathsf{E}}$, and thus we cannot invoke Lemma IV.3 to securely evaluate $\tilde{C}$ (and do not know how to obtain an efficient protocol that does). 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 $\mathbf{C}^{\mathsf{E}}$.

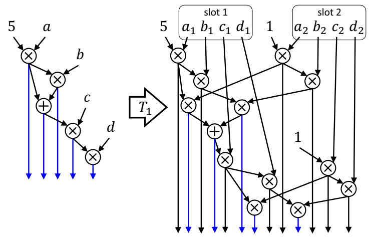
We thus take an alternative approach, which leverages the fact that C lies in the class $\mathbf{C}^{\mathsf{S}}$. Intuitively, instead of combining shares at the beginning, $\tilde{C}$ combines shares "on the fly", as the circuit is computed, as we now describe.

First consider the simple case where C has no non-trivial addition gates, i.e., all gates are either multiplication gates or addition gates that output a constant. Our reduction then outputs a circuit $\tilde{C}$ that contains n copies of C as a sub-circuit (one for each party) such that the $\#\mathsf{inputs}(C)$ inputs of each copy are assigned to a separate input slot of $\tilde{C}$; corresponding outputs of each copy are then multiplied together, thereby combining the shares, via $O(n \cdot \#\mathsf{outputs}(C))$ auxiliary multiplication gates. See Figure 3a for an example.

More generally, of course, C may include addition gates and, in such a case, the reduction is more complex,



(a) Example where C has only multiplication gates.



(b) Example where C has both addition and multiplication gates.

Fig. 3: Two examples of a circuit C in $\mathbf{C}^{\mathsf{S}}$ and the corresponding circuit $\tilde{C} := T_1(C, n)$ in $\mathbf{C}^{\mathsf{E}}$ for $n = 2$ parties. 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).

because merely individually evaluating n copies of C and then combining corresponding outputs does not compute the correct function. The reason is not surprising: while multiplicative sharing of inputs commutes with multiplication, it does not commute with addition, and thus it is hard to obtain multiplicative shares of the result of an addition. To circumvent this problem, we break the circuit down into components "separated" by additions, and apply the above idea separately to each. In-between components, before each addition, we combine shares.

In somewhat more detail, our construction works as follows. For each multiplication gate g_{mul} of C, we add to $\tilde{C}$ a sub-circuit, consisting of $O(n)$ multiplication gates, that combines a value computed so far with all the shares of g_{mul}'s right input (which is, by definition of $\mathbf{C}^{\mathsf{S}}$, an input of C). For each addition gate g_{add} of C, we add one addition gate to $\tilde{C}$ that combines the values computed so far. Crucially, each of these sub-circuits, as well as their combination, lies in $\mathbf{C}^{\mathsf{E}}$. See Figure 3b.

A notable efficiency feature of our reduction is that it ensures that the E-depth of $\tilde{C}$, which determines the number of rounds required to securely evaluate $\tilde{C}$, is "small": it is bounded above by n times the S-depth of C. Indeed, there are multiple ways to combine the aforementioned sub-circuits, but many such ways yield much worse efficiency, e.g., E-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 S-depth, this feature is critical.

The simulator in Π^{S}. The construction of $\tilde{C}$ must not only respect syntactic and efficiency requirements (e.g., lie in $\mathbf{C}^{\mathsf{E}}$, 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^{E} for $\tilde{C}$ actually gives rise (with some simple changes) to a sampling protocol Π^{S} that implements the ideal functionality of C. Since our construction of $\tilde{C}$ introduces additional, spurious outputs, the simulator in Π^{S} 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 do not leak additional information beyond the outputs that incorporate every party's share.

For an arbitrary circuit in $\mathbf{C}^{\mathsf{E}}$ 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 the evaluation protocol

We sketch the proof of Lemma IV.3. The evaluation protocol $\Pi^{\mathsf{E}} = (\Pi, V, S)$ for $C \in \mathbf{C}^{\mathsf{E}}$ works 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 4).
- In each one of the subsequent $\mathsf{depth}_{\mathsf{E}}(C)$ rounds ($t = 2, \ldots, \mathsf{depth}_{\mathsf{E}}(C) + 1$), each party i determines if there are any gates g in $\mathsf{gates}(C)$ such that (i) the E-depth of $\mathsf{output}(g)$ equals the round number minus 1 (i.e, $t - 1$), and (ii) if $\mathsf{type}(g) = \mathsf{mul}$ then $\mathsf{R\text{-}input}(g)$ depends only on inputs in $\mathsf{inputs}(C, i)$. 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 4). In this way, the parties prove correct evaluation of all gates of C, first processing

all gates whose outputs have E-depth 1, then all those whose outputs have E-depth 2, and so on.

- After $\mathsf{depth}_{\mathsf{E}}(C)$ such rounds, in the last round ($t = \mathsf{depth}_{\mathsf{E}}(C) + 2$), party 1 consults the broadcast messages so to gather, and broadcast in a single message, the encoding of the value of every output of C. The purpose of this last round is to construct a syntactically well-formed output of the protocol; tasking party 1 to do so is an arbitrary choice.

Since the circuit C belongs to the circuit class $\mathbf{C}^{\mathsf{E}}$, by definition of $\mathbf{C}^{\mathsf{E}}$, whenever a party i is supposed to prove correct evaluation of a gate g: (i) if g is an addition gate, then encodings for g's inputs have been broadcast in previous rounds, and (ii) if g is a multiplication gate, an encoding for g's left input has been broadcast in previous rounds (or computed by i in this round) and i knows the value for its right input. In either case, party i can compute an encoding for g's output, and knows a witness to the NP statement that attests to this encoding's correctness. Moreover, note that, again since C belongs to $\mathbf{C}^{\mathsf{E}}$, every gate's output wire is also an output wire of C, so that broadcasting encodings of every gate's output does not leak information beyond what is leaked by the output of the ideal functionality, $C(\vec{\sigma}) \cdot \mathcal{G}$.

The transcript of broadcast messages can be checked by a verifier V, by ensuring 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 generated by a simulator S, having access to the encoding of the circuit's output, by simulating each proof of correct evaluation.

V. Optimizations for duplex-pairing groups

The use of NIZKs in Theorem IV.1 is "light": the sampling protocol uses NIZKs for two relations, denoted $\mathscr{R}_{\mathrm{A}}$ and $\mathscr{R}_{\mathrm{B}}$ and defined in Figure 4, that involve only arithmetic in $\mathbb{G}$ and invocations of the commitment verifier $\mathsf{COMM.Ver}$. While the theorem 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; our implementation and evaluation target this special case, which occurs, e.g., in the setting of public-parameter generation for zk-SNARKs.

Strategy. The sampling protocol of Theorem IV.1 is obtained in two steps: a reduction from sampling to evaluation (Lemma IV.2), and an evaluation protocol (Lemma IV.3). The reduction is efficient, so we focus on optimizing the evaluation protocol, by suitably instantiating the commitment scheme and NIZKs for $\mathscr{R}_{\mathrm{A}}$ and $\mathscr{R}_{\mathrm{B}}$. This instantiation relies on random oracles.

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 by applying point decompression to two random strings, or heuristically, to SHA256(0) and SHA256(1)). A Pedersen commitment cm for a value x is obtained by letting cr be a random element of $\mathbb{F}_r$ and computing $\mathsf{cm} := x \cdot \mathcal{P} + \mathsf{cr} \cdot \mathcal{Q}$.

Choice of NIZK for the relation $\mathscr{R}_{\mathbf{A}}$. To prove knowledge of a committed value x encoded in a commitment cm, we use an adapted version of Schnorr's protocol [69] 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 $u := \alpha + c \cdot x$, $v := \beta + c \cdot \mathsf{cr}$. 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 [47] (in our concrete implementation, using SHA256 hashing).

Choice of NIZK for the relation $\mathscr{R}_{\mathbf{B}}$. 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 committed 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 empty 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

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 $\mathsf{COMM.Ver}(\sigma, \mathsf{cm}, \mathsf{cr}) = 1$, when parsing $\mathbb{x}$ as a commitment cm and $\mathbb{w}$ as a tuple (σ, cr) for which $\sigma \in \mathbb{F}_r$ and cr is commitment randomness.

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 $(\mathcal{R}, \mathcal{P}, \alpha, b, c)$ and $\mathbb{w}$ as a tuple (σ, cr).
2) Check that the $\mathbb{G}$-element $\mathcal{R}$ equals the $\mathbb{G}$-element $\alpha\sigma \cdot \mathcal{P}$.
3) If $b = 0$, check that $\mathsf{COMM.Ver}(\sigma, c, \mathsf{cr}) = 1$; if $b = 1$, check that the $\mathbb{G}$-element c equals the $\mathbb{G}$-element $\sigma \cdot \mathcal{G}$ (and ignore cr).

Fig. 4: Description of the two NP relations $\mathscr{R}_{\mathrm{A}}$ and $\mathscr{R}_{\mathrm{B}}$.

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 [70]. 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 $\mathbf{C}^{\mathsf{S}}$, 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 commitment schemes and NIZKs very efficiently.)

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 $\mathbf{C}^{\mathsf{S}}$. To facilitate this application, we have integrated our code with `libsnark` [45], a `C++` library for zk-SNARKs. (In particular, 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.

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

A critical issue is that C_{PGHR} and C_{DFGK} have size quasilinear in the circuit whose satisfiability is being proved. A naive implementation of the computation pattern of the zk-SNARK's generator results in circuits that are not in $\mathbf{C}^{\mathsf{S}}$; conversely, a naive implementation in $\mathbf{C}^{\mathsf{S}}$ results in circuits of quadratic size. Via careful design, quasilinear-size circuits in $\mathbf{C}^{\mathsf{S}}$ can be obtained.

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 $\mathbf{C}^{\mathsf{S}}$ (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 [71] at 128 bits of security.

Costs for the general case. Our system's efficiency only depends on the size and S-depth of the circuit C in $\mathbf{C}^{\mathsf{S}}$, and also n (the number of participating parties). In Figure 6 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.

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 S-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 $2N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 10$ and S-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 $11N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 38$ and S-depth 3.

These costs are summarized in Figure 5 .

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

- **Example #1:** the circuit C targets Zerocash [8]. 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.
- **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.

Figure 5 reports the size and S-depth of C for these two examples, and Figure 6 reports the corresponding costs.

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].

Acknowledgments

We thank Bryan Parno and the anonymous referees for numerous comments on an earlier draft that greatly improved the clarity of this paper.

This work was supported by: The Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211; the Broadcom Foundation and Tel Aviv University Authentication Initiative; the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370; the Check Point Institute for Information Security; the U.S. Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211; the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258; the Israeli Centers of Research Excellence I-CORE program (center 4/11); the Israeli Ministry of Science and Technology; the Leona M. & Harry B. Helmsley Charitable Trust; MIT Office of the Provost; The National Science Foundation under award EFRI-1441209; and the Office of Naval Research under contract N00014-11-1-0470.

Appendix A
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 $\mathbf{C}^{\mathsf{S}}$. 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 $\mathbf{C}^{\mathsf{S}}$. More precisely, we consider [25]'s zk-SNARK, which modifies [21]'s.

Supported NP relations. This zk-SNARK supports arithmetic circuit satisfiability, i.e., 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.

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

zk-SNARK	**Circuit satisfiability of D when D is**	**Circuit C in $\mathbf{C}^{\mathsf{S}}$** $\mathsf{size}(C)$	$\mathsf{depth}_{\mathsf{S}}(C)$
Danezis et al. [31]	a N_{w}-wire N_{g}-gate boolean circuit	$2N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 10$	2
Parno et al. [21], Ben-Sasson et al. [25]	a N_{w}-wire N_{g}-gate arithmetic circuit	$11N_{\mathsf{w}} + 2^{\lceil \log_2 N_{\mathsf{g}} \rceil}(\lceil \log_2 N_{\mathsf{g}} \rceil + 1) + 38$	3
Ben-Sasson et al. [8]	Example #1's arithmetic circuit	138,467,206	3
Ben-Sasson et al. [39]	Example #2's arithmetic circuit	8,027,609	6

Fig. 5: Size and S-depth of the circuit C in $\mathbf{C}^{\mathsf{S}}$ obtained from D, for various choices of D.

Complexity measure	**Cost for** **general case**	**Example #1**	**Example #2**
number of rounds	$n \cdot \mathsf{depth}_{\mathsf{S}}(C) + 3$	$3n + 3$	$6n + 6$
each party's time complexity	$0.035 \cdot \mathsf{size}(C)$ ms	14,124 s	4,048 s
number of broadcast messages	$n \cdot (\mathsf{depth}_{\mathsf{S}}(C) + 3)$	$6n$	$6n$
transcript size	$0.072 \cdot n \cdot \mathsf{size}(C)$ kB	$12{,}877 \cdot n$ MB	$906 \cdot n$ MB
transcript verification time	$1.03 \cdot n \cdot \mathsf{size}(C)$ ms	$196{,}208 \cdot n$ s	$50{,}945 \cdot n$ s

Fig. 6: Our system's costs for the general case, Example #1, and Example #2; n is the number of parties.

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.[6]

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 $\mathbf{C}^{\mathsf{S}}$.

Supported NP relations. This zk-SNARK supports boolean circuit satisfiability, i.e., 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.

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.[7]

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

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

References

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

[2] 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.

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

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

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

[8] 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.

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

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

[16] 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.

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

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

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

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

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

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

[52] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay — a secure two-party computation system," in *SSYM '04*, 2004.

[53] a. shelat and C.-h. Shen, "Fast two-party secure computation with minimal assumptions," in *CCS '13*, 2013.

[54] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *SP '13*, 2013.

[55] A. C.-C. Yao, "How to generate and exchange secrets," in *SFCS '86*, 1986.

[56] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *Journal of Cryptology*, 2009.

[57] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: a system for secure multi-party computation," in *CCS '08*, 2008.

[58] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen, "Asynchronous multiparty computation: Theory and implementation,"

in *PKC '09*, 2009.

[59] J. Katz, R. Ostrovsky, and A. Smith, "Round efficiency of multi-party computation with a dishonest majority," in *EUROCRYPT '03*. Springer-Verlag, 2003.

[60] R. Pass, "Bounded-concurrent secure multi-party computation with a dishonest majority," in *STOC '04*. ACM, 2004.

[61] C. Orlandi, "Is multiparty computation any good in practice?" in *ICASSP '11*, 2011.

[62] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - Or: Breaking the SPDZ limits," in *ESORICS '13*, 2013.

[63] I. Damgård, R. Lauritsen, and T. Toft, "An empirical study and some improvements of the MiniMac protocol for secure computation," in *SCN '14*, 2014.

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

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

[66] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold fhe," in *EUROCRYPT '12*, 2012.

[67] S. Garg, C. Gentry, S. Halevi, and M. Raykova, "Two-round secure MPC from indistinguishability obfuscation," in *TCC '14*, 2014.

[68] G. Asharov and Y. Lindell, "A full proof of the BGW protocol for perfectly-secure multiparty computation," ePrint 2011/136, 2011.

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

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

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