Homomorphic Secret Sharing with Verifiable Evaluation

Arka Rai Choudhuri[1], Aarushi Goel[2], Aditya Hegde[2], and Abhishek Jain[1,2]

[1]Nexus. arkarai.choudhuri@gmail.com
[2]Purdue University. aarushi@purdue.edu
[3]Johns Hopkins University. {ahegde,abhishek}@cs.jhu.edu
[4]NTT Research. abhishek.jain@ntt-research.com

Abstract

A homomorphic secret sharing (HSS) scheme allows a client to delegate a computation to a group of untrusted servers while achieving input privacy as long as at least one server is honest. In recent years, many HSS schemes have been constructed that have, in turn, found numerous applications to cryptography.

Prior work on HSS focuses on the setting where the servers are semi-honest. In this work we study HSS in the setting of malicious evaluators. We propose the notion of HSS *with verifiable evaluation* (ve-HSS) that guarantees correctness of output *even when all the servers are corrupted.* ve-HSS retains all the attractive features of HSS and adds the new feature of succinct public verification of output.

We present *black-box* constructions of ve-HSS by devising generic transformations for semi-honest HSS schemes (with negligible error). This provides a new non-interactive method for verifiable and private outsourcing of computation.

Contents

1 Introduction **1**
- 1.1 Our Results . . . 2
- 1.2 Applications . . . 5
- 1.3 Related Work . . . 6

2 Technical Overview **7**
- 2.1 Single Client HSS with Verifiable Evaluation . . . 7
- 2.2 Groth16 zkSNARKs [Gro16] are Splittable . . . 12
- 2.3 Splittable Zero-Knowledge BARGs . . . 14
- 2.4 Multi-Client HSS with Verifiable Evaluation . . . 17
- 2.5 Applications . . . 18

3 Preliminaries **20**
- 3.1 Homomorphic Secret Sharing . . . 20
  - 3.1.1 HSS Schemes from Prior Works . . . 23
- 3.2 Succinct Non-interactive Arguments . . . 24

4 Splittable zkSNARG **26**
- 4.1 Distributed Prover Setting . . . 28
- 4.2 The Groth16 zkSNARK . . . 30
- 4.3 The Waters-Wu zkBARG . . . 34

5 HSS with Verifiable Evaluation **41**
- 5.1 Instantiations . . . 48

6 Multi-Client HSS with Verifiable Evaluation **50**
- 6.1 Construction With Strong Privacy . . . 54
- 6.2 Simpler Construction Without Strong Privacy . . . 61
- 6.3 Instantiations . . . 62

7 Applications **63**
- 7.1 Private Delegation of Computation . . . 63
- 7.2 Collaborative zkSNARKs . . . 64

1 Introduction

In a t-private homomorphic secret sharing (HSS) scheme [BGI16a], a client uses a sharing algorithm to split a secret x into multiple shares such that any subset of at most t shares do not reveal anything about x. Such a scheme additionally supports homomorphic computation on the shares via the following algorithms:

- A local evaluation algorithm Eval that given a function f, maps a share x_i to an output share y_i.
- A reconstruction algorithm Rec that computes the output $y = f(x)$ from the output shares $\{y_i\}_i$.

Similar to additive secret sharing schemes, the reconstruction algorithm Rec is simply an additive function over the output shares. This in turn implies that the size of each share output by Eval is *succinct* and only depends on the output length of f, and is otherwise independent of the description of f.

A t-private HSS allows a client to delegate a (possibly resource-intensive) computation to a group of untrusted servers while achieving input privacy against any t number of servers.[1] This feature, in turn, has many applications in cryptography, including sublinear-communication secure multiparty computation (MPC) [BGI16a, Cou19, CM21, DIJL23], multi-server private information retrieval [GI14, BGI16a], correlated-randomness generators [BCGI18, BCG+19], reusable MPC [BGMM20], constrained pseudorandom functions [CMPR23], and more. By now, several constructions of HSS are known from standard group-based [BGI16a, BGI17, BCG+17, OSY21, RS21, ADOS22], lattice-based [DHRW16, BKS19] and code-based assumptions [BCG+19, CM21, DIJL23]. Some of these schemes differ in the number of servers they can support (two or more), the class of functions that can be evaluated, as well as whether or not they have (non-negligible) correctness error.

Prior work in this area focuses on semi-honest servers who perform the local evaluations honestly (see Section 1.3 for some exceptions). This means, in particular, that the output is only guaranteed to be correct when the servers are honest. In this work, we study HSS in the setting where the servers may be *malicious* and can deviate from the honest evaluation strategy. In particular, the new property we seek is public verifiability of the output by any external auditor, even when all the servers are malicious. Viewed from the lens of delegation of computation, we seek *verifiable* and private delegation of computation.

HSS with Verifiable Evaluation. We propose a new notion of HSS *with verifiable evaluation* (ve-HSS) that retains the attractive features of HSS and adds public verifiability to the output of the computation. Similar to (semi-honest) HSS, a ve-HSS scheme is equipped with sharing, Eval and Rec algorithms that provide the same functionality and efficiency properties. Since the evaluators may be malicious, the reconstruction algorithm Rec in ve-HSS produces a "candidate" output. We require an additional algorithm Verify that either accepts or rejects this output. Importantly, verification is *succinct*, namely, Verify runs in time independent of the complexity of the evaluated function and the number of servers.

We first augment the HSS privacy notion to require that the privacy of the client's input should hold even if the malicious servers learn whether the candidate output produced was accepted or rejected by the client. This augmentation is necessary since malicious servers may produce maliciously computed shares, and attempt to glean information based on the client behavior. As in the case of HSS, we parameterize the privacy requirements by the number of corrupt servers t.

Motivated by the application to verifiable delegation of computation, we consider the following notions of soundness:

- *Local Soundness*: This notion guarantees that for any shared input $x = (x_{\mathsf{pub}}, x_{\mathsf{priv}})$ — where x_{pub} is the public input and x_{priv} is the private input known only to the client — the Verify algorithm, given x, accepts a candidate output y if and only if y is the result of evaluating the function f on x.

[1] This is analogous to fully homomorphic encryption (FHE) which allows private delegation of computation to a single server.

– *Public Soundness*: This notion guarantees that for any shared input $x = (x_{\mathsf{pub}}, x_{\mathsf{priv}})$, the Verify algorithm, given x_{pub}, accepts a candidate output y if and only if $y = f(x_{\mathsf{pub}}, \cdot)$ for "some" private input.

Crucially, we require these soundness properties to hold *even when all the servers are corrupted.* This means that an external auditor can publicly verify the correctness of the output without knowing anything about (or placing any trust assumptions in) the entities who computed the output. This property makes our notion *stronger* than standard secure multiparty computation (MPC), which can only guarantee output correctness when at least one party is honest.

It is easy to see that ve-HSS with succinct public verification implies succinct non-interactive arguments (SNARGs) [Mic94]. Indeed, the local soundness property implies SNARGs for deterministic computations while public soundness implies SNARGs for non-deterministic computations. The class of languages supported by the SNARG is determined by the family of functions supported by the ve-HSS scheme.

Multi-Client HSS with Verifiable Evaluation. We also study verifiability for *multi-client* HSS, where a set of mutually distrusting clients can share their inputs to a set of servers who can then perform joint evaluations on the inputs of the clients. The notions of privacy and soundness in this setting are strengthened to allow for collusions between the clients and servers. In particular, local soundness now allows each client to verify whether its private input x_i was used in the computation, i.e., $y = f(\cdot, x_i, \cdot)$ even if all the remaining clients and all the servers are corrupted. Public soundness, as before, guarantees that the output was obtained by evaluating f on the fixed public input and "some" set of private inputs, i.e., $y = f(x_{\mathsf{pub}}, \cdot, \cdot)$, even if all parties are corrupt.

1.1 Our Results

In this work, we study HSS with verifiable evaluation with a focus on building solutions that make only *black-box* use of cryptography. We obtain new results both for the single-client and multi-client settings.

Single Client HSS with Verifiable Evaluation. We present two sets of results for single-client homomorphic secret sharing (HSS) with verifiable evaluation: one in the generic group model and the other in the standard model. Below, we present the results for each model separately, starting with our result in the generic group model.

Theorem 1 (Generic Group Model, Single Client, Informal)**.** *In the generic group model, we construct a single client m-server HSS with verifiable evaluation that only makes black-box use of cryptography:*

– *For* NC^1*: from (i) class groups assumptions; (ii) Decisional Composite Residuosity (DCR) assumption; or (iii) Learning with Errors (LWE) assumption.*

– *For* $\mathsf{P/poly}$*: from (i) the security of indistinguishability obfuscation (iO) + one-way functions (OWF); or (ii) security of fully homomorhpic encryption (FHE).*

In the standard model, we consider boolean circuits that are SIMD (single instruction multiple data). We obtain analogous results as in the generic group model, but in the SIMD setting.

Theorem 2 (Standard Model, Single Client, Informal)**.** *In the standard model, we construct a single client m-server HSS with verifiable evaluation that only makes black-box use of cryptography:*

– *For* SIMD NC^1*: from the subgroup decision assumption in addition to either (i) class groups assumptions; (ii) Decisional Composite Residuosity (DCR) assumption; or (iii) Learning with Errors (LWE) assumption.*

– *For* SIMD $\mathsf{P/poly}$*: from the subgroup decision assumption in addition to either (i) the security of indistinguishability obfuscation (iO) + one-way functions (OWF); or (ii) security of fully homomorphic encryption (FHE).*

Main Compiler. The results stated above for the single-client setting are based on currently known results for semi-honest HSS. However, we derive these results using a general compiler that takes as its starting point any standard semi-honest HSS scheme with negligible correctness error. This means future constructions of such HSS schemes can be plugged into our compiler to obtain new instantiations.

At a high level, our compiler adds verifiability to a semi-honest HSS scheme by requiring the servers to additionally compute a zero-knowledge succinct non-interactive arguments (zkSNARGs) of correct computation *under the hood* using the semi-honest HSS, i.e. the zkSNARG is computed in a distributed manner by all the servers. In order to ensure that the compiler only makes black-box use of cryptography, we enforce certain structural (and security) constraints on the zkSNARGs. We formalize these properties via a new notion of of zkSNARGs that we call *splittable* zkSNARGs.

We will describe the security properties shortly, but the primary 'structural' properties we require from such a splittable zkSNARG is that the computation of the zkSNARG proof can be divided into an initial *non-linear* computation phase over all of its input, followed by a computation phase that exclusively performs *linear* operations on the output of the non-linear phase. Importantly, we further require that the non-linear phase is non-cryptographic. We make this distinction since in our compiler the non-linear phase will be computed via the (semi-honest) HSS scheme, and the linear phase will be computed on the additive shares obtained from the aforementioned HSS evaluation.

With our new tool splittable zkSNARGs, we present below our main theorem statement: a compiler for transforming any semi-honest single-client HSS scheme with at most negligible correctness error into a single-client HSS with verifiable evaluation (ve-HSS).

Theorem 3 (Main Compiler, Informal)**.** *Let* HSS *be a single-client m-server HSS scheme for a class of circuits* $\mathbb{C}$ *with negligible correctness error. Then given a splittable zkSNARG for* $\mathbb{C}'$ *where the non-linear computation phase can be implemented using a circuit in* $\mathbb{C}$*, there exists a single client m-server HSS with verifiable evaluation* ve-HSS *for circuit class* $\min\{\mathbb{C}, \mathbb{C}'\}$*, that only makes black-box use of cryptography.*

Here $\min\{\mathbb{C}, \mathbb{C}'\}$ simply refers to the smaller class among the two circuit classes. We present instantiations of our theorem in the table below using single-client HSS schemes that have negligible correctness error.

Splittable zkSNARG		HSS		ve-HSS
Assumption	Circuit Class $\mathbb{C}'$ (implementable in $\mathbb{C}$)	Assumption	Circuit Class $\mathbb{C}$	Circuit Class $\min\{\mathbb{C}, \mathbb{C}'\}$
Generic Group Model [Gro16]	NC^1 (in NC^1)	LWE [BKS19]	NC^1	NC^1
		DCR [OSY21, RS21]		
		Class Groups [ADOS22]		
	P/poly (in P/poly)	iO and OWF [BGI15]	P/poly	P/poly
		FHE [DHRW16, BGI+18]		
Subgroup Decision Assumption [WW22]*	SIMD NC^1 (in NC^1)	LWE [BKS19]	NC^1	SIMD NC^1
		DCR [OSY21, RS21]		
		Class Groups [ADOS22]		
	SIMD P/poly (in P/poly)	iO and OWF [BGI15]	P/poly	SIMD P/poly
		FHE [DHRW16, BGI+18]		

Figure 1: We present our results for the single client setting, where the phrase *implementable in* $\mathbb{C}$ refers to the fact that the non-linear computation of the splittable zkSNARK is implementable by a circuit in $\mathbb{C}$. The assumption for the constructed ve-HSS scheme is the combination of the assumption for the splittable zkSNARG and the corresponding HSS scheme. [WW22]* indicates that we use our extended zkBARG to instantiate the compiler. LWE - learning with errors, DCR - decisional composite residuosity, OWF - one way functions, iO - indistinguishability obfuscation, FHE - fully homomorphic encryption.

We describe next our results for the aforementioned tool of splittable zkSNARGs, needed to instantiate our compiler.

Splittable zkSNARGs. We have already described the structural requirements for splittable SNARGs, where the computation of the proof can be split into non-linear and linear phases of computation. For security, in addition to the regular soundness properties of zkSNARGs, we require a splittable zkSNARG to have *robust verification*, i.e. the verifier will reject all proofs that contain additive errors. We defer a formal definition of this notion to later. We present two instantiations of splittable zkSNARGs that can be used to instantiate our compilers.

Spittable zkSNARGs for NP. Our first instantiation supports general NP languages and is based on the Groth16 zkSNARK[2] [Gro16] that achieve perfect zero-knowledge and computational soundness (proven in the generic group model). We show that the Groth16 zkSNARKs satisfy all the properties of a splittable zkSNARG.

Moreover, we show that the non-linear computation phase in the Groth16 zkSNARK requires evaluating a circuit of depth $O(D)$ (over a $\mathsf{poly}(\lambda)$-bit integer ring) if the relation circuit has depth D.

Spittable zkSNARGs for batch-NP. Our second instantiation supports batch-NP languages and can be based on *standard assumptions*. Recall that while SNARGs for all NP languages are not known based on standard assumption, SNARGs for batch-NP (also referred to as BARGs) [BHK17, KPY19, CJJ21] are known from standard assumptions [CJJ21, CJJ22, HJKS22, WW22, CGJ+23]. BARGs allow a prover to convince a verifier of the validity of k NP statements using a proof that is significantly smaller than the combined witness of the k statements. Our splittable zkSNARG for batch-NP is based on the scheme of [WW22] that uses composite-order pairing groups and achieves computational soundness under the subgroup-decision assumption.

We first show how to lift the BARGs of [WW22] to achieve the *perfect zero-knowledge* property. Our construction only makes *black-box* use of cryptography.

Theorem 4 (Black-box Perfect Zero-knowledge BARGs, Informal)**.** *Assuming the hardness of the subgroup decision problem in a composite-order pairing group, there exists a BARG for boolean circuit satisfiability that achieves perfect zero-knowledge and makes only black-box use of the group. The proof size is* $\mathsf{poly}(\lambda, |C|)$*, the CRS size is* $m^2 \cdot \mathsf{poly}(\lambda)$ *and the verification complexity is* $\mathsf{poly}(\lambda, |C|) + \mathsf{poly}(\lambda, m, n)$*, where* λ *is a security parameter,* $C : \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ *is the Boolean circuit,* n *is the statement size, and* m *is the number of instances.*

We note that, in general, zkBARGs can be obtained by compiling BARGs with non-interactive zero knowledge proofs. Furthermore, recent works [CW23, BKP+23, BWW23] have shown how to construct NIZKs from BARGs, which in turn, yields zkBARGs from BARGs. While these transformations are generic, they make *non-black-box* use of cryptography in several steps. Theorem 4, in contrast, yields a new and direct construction of zkBARGs based on pairing groups that makes black-box use of the group.

We additionally demonstrate that our zkBARG construction obtained in the above theorem is indeed a splittable zkSNARG for batch-NP. We shall use this to construct ve-HSS schemes that can evaluate SIMD boolean circuits. In particular, we show that when computing the zkBARG for any SIMD computation, the non-linear computation phase can be computed by a circuit of depth $O(D)$ (over a $\mathsf{poly}(\lambda)$-bit integer ring), where D is the depth of the SIMD computation.

Instantiating our compiler in Theorem 3 using (i) semi-honest HSS schemes for NC^1 (assuming DCR [OSY21, RS21] or LWE [BKS19] or class groups [ADOS22]) and $\mathsf{P/poly}$ (assuming iO + OWF [BGI15] or FHE [DHRW16, BGI+18]) ; and (ii) splittable zkSNARGs for NC^1 and $\mathsf{P/poly}$ (in the generic group model)

[2]We use zkSNARK instead of zkSNARG when the underlying scheme is also an *argument of knowledge*.

and SIMD NC^1 and SIMD P/poly (assuming hardness of the decision subgroup problem) give us our main results in Theorems 1 and 2.

Multi-Client HSS with Verifiable Evaluation. Finally, we extend our results to the *multi-client* setting. Starting from semi-honest public-key HSS schemes[3] with negligible error, we present a compiler for public-key multi-client ve-HSS using Splittable zkSNARGs and obtain the following results:

1. *Semi-Honest Clients:* When clients are semi-honest, we present a compiler for transforming any public-key HSS scheme with at most negligible correctness error into a multi-client ve-HSS using a suitable Splittable zkSNARG, assuming a secure "public-key setup".

   When instantiated with HSS schemes based on DCR [OSY21, RS21] and class group [ADOS22], our compiler yields a fully black-box solution for multi-client ve-HSS, where the public key setup can either be generated by a trusted third party or jointly by the servers using a black-box one-round maliciously secure distributed protocol from [ADOS22]. When using other HSS schemes to instantiate our compiler, the setup must be done by a trusted entity. This is because black-box protocols for public-key setup are not known for these other HSS schemes.

2. *Malicious Clients:* When the clients are malicious, we instantiate our compiler using HSS schemes based on DCR [OSY21, RS21] and class groups [ADOS22].

   If some clients are malicious, it is crucial to ensure that they send well-formed HSS shares of their respective inputs. In addition to the distributed key setup protocol from [ADOS22], in this setting, we also use the black-box, one-round, maliciously secure, input sharing protocol for these HSS schemes from [ADOS22] to instantiate our compiler.

1.2 Applications

We now discuss some applications of our ve-HSS constructions.

Private Delegation of Computation. Our notion of ve-HSS provides a black-box solution to the problem of private and verifiable delegation of computation (by a single or multiple clients) to a group of untrusted servers. Our HSS-based approach offers a significant advantage over the FHE-based approaches explored in [GGW24, ACGS23]. In the FHE-based approach, to make the final proof publicly verifiable, the client must perform some post-processing on the output sent by the server. In contrast, our approach yields a truly non-interactive solution, where the output computed by the servers is already publicly verifiable and requires no additional intervention from the client. To the best of our knowledge, this yields the first such non-interactive and black-box solution for delegating the computation of *non-cryptographic functions*.

Next, we demonstrate that our approach can also be used to delegate the computation of some *cryptographic functions*, while also being black-box in the cryptographic operations involved in the computation of these functions. Specifically, we highlight the application of our constructions to the problem of privacy-preserving delegation of zkSNARG computations in a black-box manner — an area that has recently garnered significant interest [OB22, GGJ+23, CLMZ23, LZW+24, GGW24].

Private Delegation of zkSNARK Computation. Let $\mathcal{R} = \{x, w \mid C(x, w) = 1\}$ be an NP relation defined by a circuit C. Consider a client who wishes to compute a proof for the statement x using a particular zkSNARG proof system Π. We show how if Π is a splittable zkSNARG, then the client can use our single-client ve-HSS scheme — built using Π itself — to outsource this proof computation.

A few remarks are in order: first, we note that our group-based ve-HSS schemes can only support NC^1 circuits (a limitation of known semi-honest HSS schemes). Second, the above solution offers several

[3] There are secret-key and public-key variants of HSS schemes. In the public-key variant, anyone with access to the public-key can generate shares of their secret whereas in the secret-key variant one needs to know the key to generate secret shares.

advantages over recent works [GGJ+23, CLMZ23, LZW+24, GGW24] that employ general-purpose MPC techniques (or FHE) for private-outsourcing of zkSNARGs: (i) MPC-based approaches [GGJ+23, CLMZ23, LZW+24] several rounds of the interaction between the servers, with communication growing with the size of the circuit C. In contrast, our approach necessitates *zero* interaction amongst the servers. (ii) As discussed above, unlike the FHE-based approach in [GGW24], the proof computed by the servers in our solution is already publicly verifiable and requires no additional intervention from the client. Moreover, unlike [GGW24], the final zkSNARK generated using our delegation framework does not reveal to the verifier whether it was generated by the client himself or using a delegation scheme.

Private Delegation of Collaborative zkSNARK Computation. A recent work by Ozdemir and Boneh [OB22] introduced the notion of *collaborative zkSNARKs* as a general tool for publicly auditable privacy-preserving computations. Collaborative zkSNARKs is a distributed protocol enabling a set of parties, each holding a share of an NP witness, to collectively compute a zero-knowledge succinct non-interactive argument (zkSNARG) for a given statement. Using similar ideas as in the single-client setting, our construction of multi-client ve-HSS can be used by a group of clients to delegate the generation of a *splittable zkSNARG* to a group of untrusted servers.

Ozdemir and Boneh's approach requires communication proportional to the size of the NP relation circuit. In contrast, a key advantage of our approach is that it requires *minimal interaction* between the servers, which are only needed for reconstructing the proof. The clients, meanwhile, only need to interact when computing and transmitting HSS shares of their combined witness to the servers. After this, the servers can independently compute the collaborative zkSNARG.

We refer the reader to Section 2.5, for a detailed overview of these applications.

1.3 Related Work

We provide a brief overview of related work.

Verifiable Function Secret Sharing. In a verfiable function secret sharing scheme [BGI16b, BBC+21, dCP22], the *client* may be malicious, and the servers want the guarantee that the *function shares that they received from the client are well-formed*, thus preventing any attempt by the client to learn unauthorized information about the servers' database. This is quite different from our notion of verifiability that aims to establish the *correctness of evaluation* in the face of malicious servers.

Maliciously Secure Sublinear MPC Based on HSS. A recent work by Abram, Damgård, Orlandi and Scholl [ADOS22] presents a black-box construction of maliciously-secure sublinear MPC using HSS. Specifically, they construct two-round sublinear MPC with a public-key setup from DCR-based HSS schemes [OSY21, RS21] as well as new HSS schemes based on class groups. The public-key setup can be instantiated with a one round distributed protocol (using a trusted setup for RSA parameter generation in the DCR case).

By virtue of being a maliciously-secure MPC protocol, their protocol guarantees correctness of output as long as at least one of the servers is honest. However, it does *not* imply our notion of HSS with verifiable evaluation which requires local and public soundness properties even when all parties are corrupted.

Homomorphic Secret Sharing with Verifiability. Some prior works [TLM18, TM19, TBM20, YO19, CZ21, CZ20, ZW22, Che23, HZ20, MTG22] have previously considered different notions of verifiability in homomomorphic secret sharing (under the umbrella term "verifiable HSS") for limited class of functions. The constructions proposed in [TLM18, TM19, TBM20] are only for computing a sum or product of all the clients' inputs and the clients are always assumed to be honest. While these results claim to achieve soundness (or verifiability) even when all servers are corrupt, they were later shown to be broken by [HZ20, MTG22]. None of the other results [YO19, CZ21, CZ20, ZW22, Che23] consider soundness when

all servers are corrupt. [YO19, CZ21] present HSS with verifiability of degree-d multiplications, where d depends on the number of servers. Verifiable HSS schemes in [CZ20, ZW22, Che23] are based on LWE and are designed for different degree polynomials (where the degree of the polynomial is dependent on the number of servers or the security parameter).

2 Technical Overview

As outlined in Section 1, our goal in this work is to design a ve-HSS scheme that uses cryptographic operations in a *black-box* manner. In this section, we discuss the main ideas underlying our result.

2.1 Single Client HSS with Verifiable Evaluation

We now describe the ideas underlying the construction of our *HSS with verifiable evaluation*. Throughout this discussion, we will operate in a simplified setting, assuming that there are only two servers[4]. These ideas can be easily generalized to settings with more than two servers.

Strawman Approach. A ve-HSS scheme proposes to add "verifiability" to existing HSS schemes. Within Cryptography, zero-knowledge proofs [GMR89] are the most widely used tool for adding "verifiability" to any computation. The celebrated result by Goldreich, Micali, Wigderson (GMW) [GMW87] showcased how any semi-honest multiparty computation protocol can be upgraded to withstand malicious corruptions, by requiring each participant to include a zero-knowledge proof (to attest to their honest behavior) with every message they send. Similarly, the folklore way to add verifiability to delegation of computation via FHE involves asking the server to additionally compute a zero-knowledge proof attesting that the FHE evaluation was done honestly.

This motivates the following natural (GMW-inspired) approach to adding verifiability to existing HSS constructions: the servers each compute a zero-knowledge proof alongside their outputs from the evaluation algorithm, certifying that they executed the evaluation algorithm honestly. Since the evaluation phase must be non-interactive and its output must be sublinear in the size of the function being evaluated, the servers can employ a zero-knowledge proof that is also a succinct non-interactive argument (SNARG) [Kil92, Mic94] that admits a succinct proof and sublinear verification time. This approach, however, inherently requires non-black-box use of cryptography. Specifically, since the evaluation algorithms of known HSS constructions rely heavily on cryptographic operations, it is unclear if the servers can compute a zero-knowledge SNARG (zkSNARG) to prove the honest execution of these operations without employing non-black-box techniques.

We remark that, other malicious MPC techniques [DPSZ12, GIP+14, CGH+18] involving information-theoretic MAC based checks[5] or the use of distributed zero-knowledge proofs [BBC+19, BGIN19, BGIN20, BGIN21] to verify correctness of computation are also not helpful in our setting. The soundness argument in these approaches hold only if a subset of the servers are allowed to be corrupt. In contrast, as discussed in Section 1, we demand stronger notions of soundness, that hold even if *all* servers are corrupt.

Our Starting Idea. In order to avoid such non-black-box use of the cryptography, our initial insight is to have the *servers jointly compute a single SNARG proof to prove that the computation is correct.* In particular, instead of proving that each server performed the evaluation honestly, this is a proof attesting that the reconstructed output y is indeed a correct result of evaluating function f on input x.

[4] Existing constructions of HSS [BKS19, OSY21, RS21, ADOS22, CM21] that meet our criteria of correctness (and support more than constant-depth functions, without relying on heavy tools such as iO and FHE) only support two servers. Other known construction of Multiparty HSS [DIJL23] only satisfy weak correctness (which is not sufficient for us).

[5] An information-theoretic MAC based approach is also used in the design of maliciously secure sublinear MPC based on HSS [ADOS22].

Now, we need to determine how the servers can compute such a proof. Our second insight is to *utilize HSS itself to compute this proof*. In other words, rather than generating the proofs "externally" to validate the correctness of HSS Eval, we want to use HSS Eval itself to generate the joint proof. We would like to rely on the self-authenticating property of SNARGs to prevent incorrect outputs from being accepted. Implementing this idea, however, is not straightforward:

- A naïve implementation of this approach would entail non-black-box use of the cryptographic operations involved in computing the underlying SNARG proof, contrary to our requirements.
- To prove the security of the resulting ve-HSS scheme, the simulator/ideal adversary in the ideal world must be able to abort the output if a malicious server misbehaves. The standard method to simulate aborts in the GMW paradigm is to check individual proofs sent by the corrupt parties. However, in this approach, since we no longer require servers to attach individual proofs, it is not immediately clear how one would detect and simulate aborts in the ideal world.

Note that even if we are willing to forgo the black-box requirement, implementing the aforementioned idea using non-black-box cryptographic operations of the underlying SNARGs presents a significant obstacle. This is because soundness of all known constructions of SNARGs for NP [Mic94, Gro16, GWC19, Gro10, Lip12, DFH12, GGPR13, BCI+13, BCCT13, BCC+17] are proven in idealized models such as the random oracle model (ROM), generic group model (GGM), or the algebraic group model (AGM). Employing the above non-black-box approach for SNARGs based on ROM or GGM would inevitably require non-black-box descriptions of these oracles. For SNARGs based on AGM, it remains unclear if the soundness of the SNARG computed using HSS (which also relies on cryptographic assumptions) can still be established in the AGM model, or if it requires making additional hardness assumptions.

Consequently, a black-box approach appears to be the only viable option for instantiating the above idea. However, it is not immediately clear whether existing HSS schemes can support computation of a SNARG inside them, while still being black-box in the underlying cryptographic operations. Nevertheless, this idea of computing SNARGs using HSS Eval forms the starting point of our approach, and we develop new ideas to overcome these challenges.

Splittable SNARGs. Since all SNARGs rely on cryptographic hardness assumptions, generating a SNARG proof for arithmetic relations involves two types of operations: cryptographic and non-cryptographic (i.e., simple field arithmetic). To compute the SNARG proof within HSS (while being black-box), our next insight is to make use of HSS Eval for the field arithmetic, and devise a separate method for the servers to compute the necessary cryptographic operations *outside* of HSS. However, since these two types of operations within SNARG generation may be highly intertwined, it may not be always be feasible to compute the cryptographic operations in a distributed and non-interactive manner. As such, it is unclear if it is even possible to implement this idea.

Our main contribution lies in showing that this idea is indeed feasible to implement. All we require is for the SNARG at hand to satisfy the following "split-prover" property. Namely, it must be possible to clearly divide the generation of splittable SNARG proofs into two steps:

1. *Low Depth Non-Cryptographic Operations:* The first step must only entail non-cryptographic operations over a ring. The computational depth of this step should be such that this overall computation is within the class of functions that are currently supported by existing HSS constructions.[6]
2. *Linearly Distributable Cryptographic Operations:* The second step of the computation may involve cryptographic operations. However, these operations must be feasible to execute in a distributed manner

[6] Most existing HSS schemes support evaluating a limited class of circuits e.g., branching programs [BGI16a, BGI17, BCG+17, DIJL23, FGJS17, BKS19, OSY21, RS21, ADOS22] and low-degree multi-variate polynomials [BCG+19, COS+22, CM21, LMS18, ILM21, FIKW22, DIJL23].

without requiring interaction. Specifically, considering that the first step is computed using HSS Eval, the servers will possess additive shares of the output from this phase. Using these additive shares, each server should be capable of independently performing certain operations and generating a partial SNARG proof. We will sometimes refer to these partial proofs as shares of the proof. Importantly, there must exist an efficient and simple algorithm to reconstruct the final proof using the partial proofs computed by all servers.

We refer to such SNARGs that satisfy these two properties as *splittable SNARGs*. Looking ahead, we will also require splittable SNARGs to satisfy a third property - a weaker form of the well-studied notion called *unambiguity* [RRR21]. We discuss this in more detail towards the end of this subsection. In subsequent subsections, we demonstrate how some existing SNARGs do satisfy these properties.

Candidate Approach. Given a splittable SNARG, as described above, and any HSS scheme with negligible correctness error, the servers run the evaluation algorithm in our ve-HSS scheme as follows:

- Given HSS shares of the input (say $x = (x_{\mathsf{priv}}, x_{\mathsf{pub}})$), the servers execute the evaluation algorithm of the underlying HSS scheme (say HSS.Eval) to compute additive shares of $y = f(x)$.
- Let $\mathcal{R}_{\mathsf{local}} : \{(f, x, y) \mid \text{st, } f(x) = y\}$ be a deterministic relation, i.e., $\mathcal{R}_{\mathsf{local}} \in \mathsf{P}$. Using HSS shares of x, the servers use HSS Eval to compute additive shares of the output of the first part of the computation (i.e., involving low depth non-cryptographic operations) required for generating a SNARG for the relation $\mathcal{R}_{\mathsf{local}}$[7].
- Using additive shares of x, y and additive shares of the outcome of the previous computation, the servers then perform the second step of the computation (i.e., linearly distributable cryptographic operations) in SNARG generation to obtain shares of the final SNARG proof.
- The servers send additive shares of y as well as shares of the SNARG proof to the client.

Finally, the client can reconstruct these shares, verify the proof (say π_{local}) w.r.t., x, y and accept y as the correct output only if this proof is successfully verified. It is easy to see that if the servers do not have access to the shares of the output or the proof and do not learn whether or not this proof verified, this approach trivially achieves privacy against malicious servers. This is because, in this non-interactive protocol, the view of the malicious server only consists of the HSS share of x that was provided to it in the input sharing phase. From the privacy guarantee of the underlying HSS scheme, it follows that this share computationally hides the input x. Moreover, local soundness of this construction follows from the soundness of the underlying SNARG.

Public Soundness. While the above approach suffices for local soundness, as previously discussed, we require the vHSS scheme to also satisfy a notion of public soundness. Moreoever, we require a stronger notion of privacy, where the servers can learn the output as well as the outcome of the honest client's verification. Recall that public soundness necessitates that upon observing the output of the evaluation phase, anyone (not solely the client) can be convinced that there exists some x_{local} such that y is a valid output of f given public input x_{pub}. To accomplish this, we can modify the aforementioned protocol as follows: in addition to providing a "local proof" Π_{priv} for the client, as described previously, the servers will also compute a "publicly verifiable SNARG" (say Π_{pub}) for the relation $\mathcal{R}_{\mathsf{pub}} = \{(f, x_{\mathsf{pub}}, y) \mid \exists x_{\mathsf{priv}}, \text{ s.t., } f(x_{\mathsf{pub}}, x_{\mathsf{priv}}) = y\}$ in a similar manner. Shares of Π_{local} will only be sent to the clients, while the shares of the output y and this publicly verifiable proof Π_{pub} will be published publicly. The client will accept the output if and only

[7]Note that given x, since the computation of $f(x)$ is deterministic, we do not need to start with HSS shares of y for this computation. It can be assumed to be implicitly computed as part of this computation.

if both Π_{local} and Π_{pub} verify. Note that unlike $\mathcal{R}_{\mathsf{local}}$, $\mathcal{R}_{\mathsf{pub}}$ is not in P. Therefore, while it suffices to use a SNARG for P for the computation of Π_{local}, we crucially require a SNARG for NP for the computation of Π_{pub}.

Privacy. Recall that our privacy notion requires that the view of the malicious server, consisting of its input share and the decision of the Verify algorithm, can be simulated solely based on the function output without the client's input. This mirrors the requirement in any MPC protocol achieving security with abort. This strong privacy guarantee is crucial to prevent an adversary from causing selective aborts based on x_{priv} (see [LP11], for a detailed discussion on this issue in MPC protocols). An observant reader might notice that achieving this notion of privacy is not as straightforward as before. This presents the following challenges:

Challenge 1: Since we allow for malicious corruptions of one of the servers, the shares of the candidate y and Π_{pub} computed in the evaluation phase are not guaranteed to have been honestly computed.[8] This *incorrect* output and proof could potentially leak information about the private input.

Addressing this Challenge: We observe that unlike in an interactive protocol, where the computation performed by the honest parties can be influenced by the strategy adopted by the adversary, this is not feasible in a non-interactive protocol. Since our evaluation phase in a non-interactive, the shares of y and Π_{pub} output by the honest party, cannot be influenced by the adversary. Therefore, an incorrect output y—which can only result from the adversary sending an incorrect share of y—does not compromise privacy.

Furthermore, since Π_{pub} is publicly revealed, ensuring privacy necessitates the use of a zero-knowledge SNARG (zkSNARG) to generate Π_{pub}. We assume that the randomness used to generate the zkSNARG Π_{pub} is sampled and HSS-shared by the client with the servers in the input sharing phase. Given HSS shares of this randomness, the computation carried out by the servers in the evaluation phase is deterministic.

Challenge 2: A rushing adversary might opt to wait for the honest server to publish their additive share of y and Π_{pub}, and then adjust their output shares of y, Π_{local} and Π_{pub}, accordingly. To prove security, we require the simulator/ideal adversary in the ideal world to be able to efficiently simulate the joint view of the adversarial server and the output of the honest client. However, if the adversary chooses to transmit malformed shares of the proofs Π_{local}, Π_{pub} (while still providing an honestly computed share of y), it becomes unclear how the simulator in the ideal world can accurately predict whether the resulting reconstructed proofs will verify with respect to x_{priv}, x_{pub} and an honestly computed y (i.e., it cannot simulate the output of the honest client).

Note that since Π_{pub} is publicly reconstructed, the simulator in the ideal world can reconstruct this proof on its own and check whether it verifies w.r.t. the reconstructed y. However, since it does not have x_{local} (i.e., the input of the honest client), the same strategy cannot be used for deciding the outcome of Π_{local}, which is verified w.r.t. x_{local}.

Addressing this Challenge: As discussed earlier, this issue does not arise in a GMW-style approach [GMW87], where each server provides an individual proof confirming honest behavior. This is because each of these proofs can be independently verified with respect to a *public* state. In our setting however, we have a jointly computed proof that needs to be verified w.r.t. a *private* state. Nevertheless, in order to simulate the outcome of Verify, we would like our simulator to perform a similar local verification based on the proof share of the corrupt sever.

To enable this, we require the splittable zkSNARG used for generating Π_{local} to satisfy an additional property, which we term *distributed-prover robust verification (DP-robust verification)*. Informally speaking, this property dictates that given a fixed (and honestly generated) share of Π_{local} computed by the honest

[8] Since our evaluation phase is non-interactive, the servers have no way to verify that this output and proof were honestly generated before reconstructing them.

server (using HSS shares of x_{priv}), a computationally bounded malicious server can (w.h.p.) only find a unique share of Π_{local}, such that the reconstructed Π_{local} verifies with respect to x_{priv}, y and x_{pub}. Using this property, the simulator in the security argument can determine the output of the honest client based on whether (1) Π_{pub} verifies; and (2) the share of Π_{local} provided by the adversarial server matches the one it should have computed given its share of x_{priv}.

Summary. We now summarize how we prove security of our proposed construction. As discussed earlier, local and public soundness follow from the soundness of the underlying zkSNARGs. To prove privacy, our simulator in the ideal world will proceed as follows:

1. The simulator queries the ideal functionality to learn y. In this non-interactive protocol, the adversary can always learn the correct output.
2. It computes HSS shares of $x_{\mathsf{priv}} = 0$ and some randomness to be used for generating π_{pub}. Indistinguishability of these shares and the honestly generated shares sent by an honest client in the real world follows from privacy of the underlying HSS scheme.
3. It uses the simulator of the underlying zkSNARG to simulate an accepting proof Π_{pub}.
4. The simulator computes the "expected" shares of y (say $y^{\mathcal{A}}$), Π_{local} (say $\Pi^{\mathcal{A}}_{\mathsf{local}}$) and Π_{pub} (say $\Pi^{\mathcal{A}}_{\mathsf{pub}}$) using HSS Eval and the HSS shares sent to the adversary.
5. It simulates the shares of y, Π_{pub} sent by the honest server as follows: $y^{\mathcal{H}} = y - y^{\mathcal{A}}$ and $\Pi^{\mathcal{H}}_{\mathsf{pub}} = \Pi_{\mathsf{pub}} - \Pi^{\mathcal{A}}_{\mathsf{pub}}$. It is easy to see that $y^{\mathcal{H}}$ is identical to the share sent by an honest server in the real world. Indistinguishability of $\Pi^{\mathcal{H}}_{\mathsf{pub}}$ from that sent by the honest server in the real world follows from zero-knowledge of the underlying zkSNARG.
6. Finally, upon receiving the shares $\bar{y}^{\mathcal{A}}, \bar{\Pi}^{\mathcal{A}}_{\mathsf{local}}, \bar{\Pi}^{\mathcal{A}}_{\mathsf{pub}}$ sent by the adversarial server, the simulator checks:
   - If $y^{\mathcal{A}} \neq \bar{y}^{\mathcal{A}}$, then from soundness of the underlying zkSNARGs, it follows that the adversary will not be able to produce shares such that the local proof verifies. And therefore the output of the honest client in this case must be $\perp$.
   - If $y^{\mathcal{A}} = \bar{y}^{\mathcal{A}}$, but $\bar{\Pi}^{\mathcal{A}}_{\mathsf{local}} \neq \Pi^{\mathcal{A}}_{\mathsf{local}}$, then from DP-robust verification of the underlying zkSNARG, it follows that the output of the honest client in this case must be $\perp$.
   - If $y^{\mathcal{A}} = \bar{y}^{\mathcal{A}}$ and $\bar{\Pi}^{\mathcal{A}}_{\mathsf{local}} = \Pi^{\mathcal{A}}_{\mathsf{local}}$, then the simulator checks if $\bar{\Pi}^{\mathcal{A}}_{\mathsf{pub}} + \Pi^{\mathcal{H}}_{\mathsf{pub}}$ is an accepting proof. If so, the output of the honest clients must be y, else it must be $\perp$.

   This concludes a proof sketch for our proposed construction.

DP-Robust Verification. A few remarks on DP-robust verification are in order:

- *Robust verification implies DP-robust verification:* Looking ahead in Section 4, we first introduce a notion called *robust verification* (a single-prover analogue of DP-robust verification) and then demonstrate that this notion implies DP-robust verification. At a high level, we say that a zkSNARG satisfies robust verification if, given a valid statement x and witness w of choice, an adversary cannot, with high probability, find a non-zero additive error ϵ such that, if Π is an honestly generated accepting proof for x, then $\Pi + \epsilon$ is also an accepting proof for x.
- *Robust Verification vs Unambiguity:* It is worth noting that robust verification represents a weaker variant of a well-studied notion [RRR21] called unambiguity of a proof system, which demands that a computationally bounded adversary cannot produce an accepting proof different from the "true/prescribed

proof" even for true statements. This notion has recently found application in connection to PPAD hardness [CHK+19a, CHK+19b, EFKP20, KPY20, JKKZ21, LV20, BCH+22]. However, achieving this property is challenging and most proof systems (including the ones we will instantiate our approach with) do not satisfy this property. Fortunately, we only require a weaker variant (as described above) of this property. It is easy to see that a proof system that satisfies the standard notion of unambiguity will also satisfy robust verification. However, the inverse implication does not hold true.

- *Native Robust verification in randomized zkSNARGs:* We are only able to prove robust verification natively for randomized variants of the splittable zkSNARGs that we use to instantiate our approach. Therefore, even though Π_{local} could potentially have been generated using a deterministic-prover zkSNARG, we will compute it using a randomized-prover zkSNARG.

- *Instantiations of Splittable zkSNARGs:* In Section 4, we show that all linear PCP-based zkSNARGs [BCI+13] satisfy the structural properties of splittable zkSNARGs. Specifically, the prover algorithm in these zkSNARGs can be divided into two steps: low-depth non-cryptographic operations and linearly distributable cryptographic operations. Moreover, it is always possible to transform a linear PCP-based zkSNARG to one that has robust verification,[9] at the cost of requiring the verifier to store private state generated during setup i.e., the resulting zkSNARG is designated-verifier. While a designated-verifier zkSNARG is compatible with our ve-HSS compiler, we show that the linear-PCP based zkSNARG proposed by Groth [Gro16] satisfies robust verification without any modifications. Furthermore, we present a zero-knowledge variant of the Waters-Wu [WW22] BARG scheme, and demonstrate that it satisfies all the structural properties of splittable zkSNARGs and has robust verification. Unlike the generic transformation, these zkSNARGs satisfy robust verification while being publicly verifiable, which may be a property of independent interest.

2.2 Groth16 zkSNARKs [Gro16] are Splittable

[Gro16] (henceforth referred to as the Groth16 zkSNARK) is a constant-sized zkSNARK, wherein the final proof consists of only three group elements. This scheme has perfect zero-knowledge, while its soundness holds in the generic group model (GGM). We observe that this scheme satisfies robust verification and all the other properties that we require from a splittable zkSNARK. In this section, we briefly recall this scheme and then discuss why it can be used to instantiate our approach.

Overview of Groth16 zkSNARKs. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T)$ be bilinear groups such that $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p with generators g_1, g_2, g_T respectively, that satisfy the relation $e(g_1, g_2) = g_T$.

- *Correlated Random Setup (CRS):* The CRS is highly structured and is relation-specific. It comprises of correlated group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.

- *Proof Generation:* The relation being proven is encoded as an instance of Quadratic Arithmetic Program (QAP) [GGPR13]. The proof generation algorithm comprises of the following steps:

  1. *Extended Witness Generation:* The initial step involves deriving an "extended witness" by using the statement and witness for the given relation. This extended witness can essentially be perceived as the complete computation trace or the list of all intermediate wire values computed during the evaluation of the relation circuit using the statement and witness as input.
  2. *Additional Constant-Depth Field Operations:* The proof generation algorithm requires performing a constant-depth computation over this extended witness (and some additional randomness sampled by the prover to ensure zero-knowledge).

[9] We thank the anonymous TCC 2024 reviewer for pointing this out.

3. *Combining with CRS Terms:* The output of the previous step is then used to linearly combine (using group exponentiations) the group elements from the CRS to compute the final proof, which comprises of three group elements: $A, C \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

– *Verification:* Given the CRS, statement and the proof elements A, B, C, the verifier performs a constant number of pairing operations to verify if the proof is accepting or not.

Splittable zkSNARG Properties. Recall that, in our application, the relation that will be proved using this zkSNARK is of the form: $\{f, x_{\mathsf{pub}}, x_{\mathsf{priv}}, y \mid \text{st.}, \ f(x_{\mathsf{pub}}, x_{\mathsf{priv}}) = y\}$, where f, y are always part of the statement and depending on the type of proof (i.e., local/public), x_{priv} is either part of the statement or the witness.

The depth of the computation necessary for generating each element of the extended witness for such relations is less than or equal to the depth of the circuit representation of f. Since the additional computation over the extended witness requires constant depth field operations, overall, the depth of the non-cryptographic operations involved in the generation of Groth16 zkSNARKs is asymptotically equivalent to that of f. However, most existing HSS schemes only support evaluating a restricted class of circuits, called Restricted Multiplication Straight-line (RMS) programs. Informally, an RMS program is an arithmetic circuit over bounded integers, where one of the two inputs of a multiplication gate must always be a circuit input wire. We show in Theorem 9 that for any function f that can be computed by an NC^1 circuit, an HSS scheme for RMS programs[10] can compute the non-cryptographic operations involved in Groth16. We refer the reader to Section 5 for more details.

The only cryptographic operations involved in the generation of this SNARK are the group exponentiations required when combing the CRS terms with the output of the non-cryptographic operations. We now explain (using a simplified example) how the servers can generate shares of A, B, C: Let $a_1, \ldots, a_\ell \in \mathbb{Z}_p$ be some terms output by the non-cryptographic operations in the previous step, and $Q_1, \ldots, Q_\ell \in \mathbb{G}_1$ be some terms in the CRS. In Groth16 zkSNARKs, the proof term A is computes as $A = \prod_1^\ell Q_i^{a_i}$. Given additive shares of $a_1, \ldots, a_\ell$ over $\mathbb{Z}_p$, the jth server can compute $A^j = \prod_1^\ell Q_i^{a_i^j}$, where a_i^j is the share of a_i held by the jth server (for each $j \in \{1, 2\}$). The final A can be reconstructed as $A = A^1 \cdot A^2$ using the shares computed by each of the servers[11]. Shares of the other two terms B and C can also be computed in a similar manner. This idea of computing group exponentiations in a distributed manner using additive shares of the exponent has been explored in several prior works (see [OB22] and references contained therein for more details).

Robust Verification. Given the above discussion, it is clear that Groth16 zkSNARK meets all the structural requirements we demand from splittable zkSNARG. To establish that this proof system is applicable within our framework, all that remains is to demonstrate that it also achieves robust verification.

Given the proof terms A, B, C, and the statement $\mathsf{st} := (x = (x_{\mathsf{priv}}, x_{\mathsf{pub}}), y)$, the verifier in Groth16 checks the following equation:

$$e(A, B) \stackrel{?}{=} e(Q_1, Q_2) \cdot e(Q_3^{x_{\mathsf{priv}}} \cdot Q_4^{x_{\mathsf{pub}}} \cdot Q_5^{y}, Q_6) \cdot e(C, Q_7)$$

where, $Q_1, \ldots, Q_7$ correspond to some group elements in the CRS that are used for this verification check.

To establish robust verification, we want to show that given some errors $g_1^{\epsilon_A}, g_2^{\epsilon_B}, g_1^{\epsilon_C}$ by the adversary, if $\epsilon_A \neq 0$ or $\epsilon_B \neq 0$ or $\epsilon_C \neq 0$, and (A, B, C) is an accepting proof, then the following is not an accepting proof (w.h.p):

$$\bar{A} = A \cdot g_1^{\epsilon_A} \quad \bar{B} = B \cdot g_2^{\epsilon_B} \quad \bar{C} = C \cdot g_1^{\epsilon_C}$$

[10] We require the HSS scheme to support evaluating RMS programs over exponentially bounded integers. This is true for all HSS schemes with negliglble correctness error.

[11] Note that this multiplicative reconstruction can also be viewed as additive reconstruction over $\mathbb{Z}_p$ in the exponent of g_1.

The verification check given these malformed proof terms can now be written as

$$
\begin{aligned}
& e(A \cdot g_1^{\epsilon_A}, B \cdot g_2^{\epsilon_B}) \stackrel{?}{=} e(Q_1, Q_2) \cdot e(Q_3^{x_{\text{priv}}} \cdot Q_4^{x_{\text{pub}}} \cdot Q_5^{y}, Q_6) \cdot e(C \cdot g_1^{\epsilon_C}, Q_7) \\
\Rightarrow\ & e(A, B) \cdot e(g_1^{\epsilon_A}, B) \cdot e(A, g_2^{\epsilon_B}) \cdot e(g_1^{\epsilon_A}, g_2^{\epsilon_B}) \\
& \stackrel{?}{=} e(Q_1, Q_2) \cdot e(Q_3^{x_{\text{priv}}} \cdot Q_4^{x_{\text{pub}}} \cdot Q_5^{y}, Q_6) \cdot e(C, Q_7) \cdot e(g_1^{\epsilon_C}, Q_7) \\
\Rightarrow\ & e(g_1^{\epsilon_A}, B) \cdot e(A, g_2^{\epsilon_B}) \cdot e(g_1^{\epsilon_A}, g_2^{\epsilon_B}) \stackrel{?}{=} e(g_1^{\epsilon_C}, Q_7) \\
\Rightarrow\ & e(g_1^{\epsilon_A}, B) \cdot e(A, g_2^{\epsilon_B}) \stackrel{?}{=} e(g_1^{\epsilon_C}, Q_7) \cdot e(g_1^{\epsilon_A}, g_2^{\epsilon_B})^{-1}
\end{aligned}
$$

We observe that because of the randomness used in the generation of A, B is uniformly sampled from $\mathbb{Z}_p$, these proof terms A and B are uniformly distributed over $\mathbb{G}_1$ and $\mathbb{G}_2$. Moreover, we know that Q_7 is guaranteed to not be g_2^0. Therefore, the probability that an adversarial server who does not have the randomness used in the generation of A and B can find non-zero field elements $\epsilon_A, \epsilon_B, \epsilon_C$ such that the above check verifies, is exponentially small (in the size of the field). As a result, this randomized variant of Groth16 satisfies robust verification. We refer the reader to Theorem 9 for a formal proof for why Groth16 zkSNARKs satisfy robust verification.

Recall that in the single-client scenario, for local proof we only need to prove a relation in P. Moreover, since this proof is only sent to the client, one might intuitively consider using a de-randomized variant of Groth16 zkSNARKs. However, given that our proof of robust verification only holds for the randomized version of Groth16, we must use this variant for local proofs as well. Overall, instantiating our approach from Section 2.1 using the Groth16 zkSNARKs gives us a single-client ve-HSS scheme. Since soundness of this proof system holds in the generic group model, our resulting ve-HSS scheme is also secure in the generic group model.

2.3 Splittable Zero-Knowledge BARGs

BARGs (Succinct Non-Interactive Batch Arguments) are SNARGs designed for a specific set of relations where a batch of statements, all requiring verification against the same NP relation, are involved. Thanks to recent advancements [CJJ21, CJJ22, HJKS22, WW22, CGJ+23], BARGs are now known from various standard assumptions, (i.e., without relying on any idealized models). As discussed in Section 1.1, [WW22] (referred to hereafter as Waters-Wu BARGs) are the only known BARGs that avoid non-black-box usage of cryptography, but they do not achieve zero-knowledge. In this section, we discuss how we can adapt the Waters-Wu BARGs to obtain a black-box construction of zero-knowledge BARG (zkBARGs). We then proceed to show that this zkBARG is a splittable zkSNARG and can be used to instantiate our approach.

Recap of Waters-Wu BARGs. We start by recalling the design of Waters-Wu BARGs. At a high level, this BARG scheme is a *commit-and-prove* SNARG that achieves non-adaptive soundness. Let $\mathbb{G}$ and $\mathbb{G}_T$ be symmetric composite order pairing groups of order $N = pq$ for primes p and q. Thus $\mathbb{G} \cong \mathbb{G}_p \times \mathbb{G}_q$ where $\mathbb{G}_p$ is a subgroup of order p generated by $g_p = g^q$ where g is the generator for $\mathbb{G}$, and $\mathbb{G}_p$ is a subgroup of order q generated by $g_q = g^p$. Further, there exists an efficiently computable non-degenerate bilinear map, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, such that for all $a, b \in \mathbb{Z}_N$ $e(g^a, g^b) = e(g, g)^{ab}$. The subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$ themselves are orthogonal with respect to the pairing operation, i.e. $e(g_p, g_q) = g_T^0$ where g_T is the generator for $\mathbb{G}_T$. The security of the construction relies on the *decision subgroup assumption* [BGN05] which states that a random element from the subgroup $\mathbb{G}_p$ is indistinguishable from a random element in the full group $\mathbb{G}$.

Let $C : \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be the Boolean circuit (composed of only NAND gates) used in the BARG, where the BARG proof is constructed over a batch of T instances $(x_1, \ldots, x_T)$ such that for each

$i \in [T]$, there exists witness w_i such that $C(x_i, w_i) = 1$. The CRS is defined to be

$$\mathsf{crs} = \left(\{A_i = g_p^{\alpha_i}\}_{i\in[T]}, A = \prod_{i\in[T]} A_i = g_p^{\sum_{i\in[T]}\alpha_i}, \{B_{i,j} = g_p^{\alpha_i\alpha_j}\}_{i,j\neq i}\right)$$

where the α_is are independently and randomly sampled from $\mathbb{Z}_N$.

- *Algebraic Vector Commitment:* For each wire k in the circuit C, a commitment U_k is computed as $U_k = \prod_{i\in[T]} A_i^{w_{i,k}} \in \mathbb{G}_p$ where $w_{i,k}$ is the k-th wire value in the i-th instance. The number of such commitments produced is equal to the number of wires in the circuit.
- *Wire Validity Check:* To prove that each commitment commits to a binary vector, the prover computes for each wire k, a proof term $V_k = \prod_{i\neq j} B_{i,j}^{(1-w_{i,k})w_{j,k}}$.
- *Gate Validity Check:* To prove that the (NAND) gate computation was done correctly, for every gate ℓ indexed by the two incoming wires k_1 and k_2 and outgoing wire k_3, the prover computes $W_\ell = \prod_{i\neq j} B_{i,j}^{w_{j,k_3} - w_{i,k_1} w_{j,k_2} - 1}$.
- *Verification:* Given crs, the commitments $\{U_k\}_k$, the wire validity proofs $\{V_k\}_k$ and gate validity proofs $\{W_\ell\}_\ell$. The verifier checks for each k, if $e(A, U_k) = e(U_k, U_k) \cdot e(g_p, V_k)$; and for each gate tuple $\ell = (k_1, k_2, k_3)$, if $e(A, U_{k_3}) \cdot e(U_{k_1}, U_{k_2}) = e(A, A) \cdot e(g_p, W)$. Additionally, the verifier re-computes the commitments for the input wires corresponding to the statements to check if they were honestly generated.

Adding Zero-Knowledge. The algebraic vector commitments to internal wire values in this scheme are deterministically computed and hence are not *hiding*, it is easy to see that as such these BARGs do not achieve zero-knowledge. Towards adding zero-knowledge to this construction, our first idea is to use a hiding commitment. Such commitments can be obtained by simply adding randomness to the above algebraic vector commitments. In particular, for each wire k in the circuit C, the prover first samples a random $r_k \in \mathbb{Z}_p$ and then computes $U_k = g_p^{r_k} \cdot \prod_{i\in[T]} A_i^{w_{i,k}} \in \mathbb{G}_p$.[12] This randomized U_k now information-theoretically hides the $w_{i,k}$ values.

For *completeness*, we now also need to modify V_k and W_ℓ, in such a way that they can somehow be verified w.r.t. the randomized commitments U_k. Importantly, these modification should not break soundness of this proof system. We propose to modify V_k as follows:

$$V_k = \frac{\prod_{i\neq j} B_{i,j}^{(1-w_{i,k})w_{j,k}} \cdot \prod_{i\in[1,m]} A_i^{r_k(1-2w_{i,k})}}{g_p^{r_k^2}}.$$

Interestingly, in Theorem 10, we show that given the randomized commitments U_k and these V_k terms, we can continue to use the same verification check as in Waters-Wu BARGs for wire validity checks. In Section 4.3, we show that W_ℓ can also be modified in a similar manner, such that, together with the randomized U_k, gate validity checks can be performed in a similar manner as in Waters-Wu BARGs. While these modifications suffice for completeness, we still need to show the resulting BARG achieves soundness and zero-knowledge.

Zero-Knowledge: Next, we show that our modified BARG achieves perfect zero-knowledge. Our first observation here is that, each U_k is identically distributed to an element that is obtained by first unformly

[12] Since the statement in public and need not be hidden, the prover can simply set $r_k = 0$ when computing commitments for the input wires corresponding to the statements.

sampling a random value (say z_k) from $\mathbb{Z}_N$ and then computing $g_p^{z_k}$. Moreover, given these values of U_k and the crs, the remaining proof terms V_k and W_ℓ are uniquely determined by the verification equations for wire-validity and gate-validity checks. Our proof for perfect zero-knowledge, now follows from these observations in a straightforward manner. Formally speaking, a simulator with knowledge of all the α_i terms used for generation of the crs can simulate an accepting proof for any batch of statements as follows: (1) for each wire k, it first samples a random $z_k \in \mathbb{Z}_N$ and then computes $U_k = g_p^{z_k}$. (2) Using knowledge of α_is and the z_k values, it can now compute all the V_k and W_ℓ terms such that all the verification checks go through. It is easy to see that this proof is perfectly indistinguishable from an honestly generation proof for a true batch of statements.

Soundness: Observe that, our idea for adding zero-knowledge to the Waters-Wu BARGs only entails making modifications to the proof generation algorithm. Importantly, we do not require modifying the crs and the verification algorithm. In other words, both the crs and the verification algorithm in this modified construction are identical to that in Waters-Wu BARGs. As a result, soundness of our scheme simply follows from the soundness of Waters-Wu BARGs.

Splittable zkSNARG Properties. Similar to the Groth16 zkSNARKs, the first step in generating our zkBARG, involves computing all wire values corresponding to the statements, followed by the computation of all the exponent terms needed to compute V_k and W_ℓ. For instance, for each V_k, these exponent terms are $(1 - w_{i,k})w_{j,k}$, $r_k(1 - 2w_{i,k})$ and r_k^2. Since these only involve operations over $\mathbb{Z}_N$, they can be computed in a distributed manner using HSS Eval.

Given the crs terms, all wire values, and these exponent terms, the subsequent step is to compute U_k, V_k, W_ℓ using group exponentiations. As discussed in Section 2.2, these operations can be performed in a non-interactive distributed manner, provided additive shares of the wire values and the exponent terms are available (which we obtain from the previous step). This establishes that our zkBARG satisfies all the structural properties of a splittable zkSNARG.

Robust Verification. Finally, we show via a sequence of the following claims that this scheme also has robust verification:

- Firstly, we observe that an adversary cannot add a non-zero error to the *vector commitments corresponding to the statement.* Recall that, since the statements are public, we do not randomize these commitments in our zkBARG. Similar to Water-Wu BARGs, the verifier verifies the correctness of these commitments by recomputing them. Therefore, any deviation from an honestly computed commitment to the statements will result in the proof being rejected.

- Next, we establish that the probability of an adversary successfully choosing a non-zero error on any of the remaining commitments U_k or the V_k terms, while still surpassing the wire-validity checks, is exponentially small. The proof of this claim employs a similar argument to our proof of robust verification for Groth16 zkSNARKs and crucially relies on the fact that the randomness r_k used in the computation of U_k remains unknown to the adversary.

- Lastly, if no errors are introduced on the U_k terms, then there exist unique values of W_ℓ that will allow the gate-validity checks to pass. This ensures that if the adversary sends errors for any of the W_ℓ terms, the verification will fail.

We defer a formal proof to Theorem 11. Instantiating our approach from Section 2.1 using this new construction of zkBARGs gives us a single-client ve-HSS scheme for SIMD computations. The security of this scheme holds in the standard model.

2.4 Multi-Client HSS with Verifiable Evaluation

So far, our discussion has been limited to the single-client setting. Now, we shift our focus to the scenario where multiple (say n) mutually distrustful clients seek to share their inputs via HSS with the servers to compute a joint function on these inputs. In this section, we now consider the setting where multiple mutually distrusting clients (we consider both consider both semi-honest and malicious client settings) have inputs that they wish to HSS share with the servers to have them compute a joint function on these inputs.

Candidate Approach. In the single-client setting, the local proof assures the client that the output was honestly computed using their input. In the multi-client setting, we require the same assurance for every client who shares their inputs via HSS with the servers. A straightforward way to achieve this would be to have the servers compute multiple local proofs, one for each server. Specifically, the local proof sent to the ith server will pertain to the relation $\mathcal{R}_i = \{x_i, y, f, x_{\text{pub}} : \exists \{x_j\}_{j=1, i\neq j}^{n}, \text{st.}, f(x_{\text{pub}}, x_1, \ldots, x_n) = y\}$. Observe that, unlike in the single-client setting, this is no longer a deterministic relation. To ensure privacy (of the other clients' inputs), we must use a zero-knowledge SNARG to generate this proof. The publicly verifiable proofs can be computed using any splittable zkSNARG as in the single-client setting, albeit for the relation $\mathcal{R}_{\text{pub}} = \{y, f, x_{\text{pub}} : \exists \{x_i\}_{i=1}^{n}, \text{st.}, f(x_{\text{pub}}, x_1, \ldots, x_n) = y\}$. Together these two types of proofs ensure local and public soundness in the multi-client setting. However, privacy of this candidate approach does not follow as easily as before.

Challenges in the Multi-Client Setting. Unlike in the single-client setting, corrupt clients can also have inputs in the multi-client setting. This presents some unique challenges in this setting.

- *Privacy:* In the single-client setting, the output y is uniquely determined by the input of the honest client. As a result, if the local proof verifies, then the correctness of the output y follows from the soundness of the underlying splittable zkSNARG. However, in the multi-client setting, the output y might not be determined by the input of any single client. Since the statements proved using the local and public proofs consist of only a subset of the inputs, we can no longer rely solely on these proofs to verify the correctness of the reconstructed output.

  We explain this using a simple example. Let us assume that we have two clients $\mathsf{C}_0, \mathsf{C}_1$ with one bit input each (let C_i have input bit b_i), who wish to delegate the computation of the product of their inputs using ve-HSS. Let us further assume that the malicious server employs the following strategy: it first waits for the honest server to send its share of the output and then chooses its share of the output adaptively, such that the reconstructed output is always 0. Now consider the scenario, where both b_0 and b_1 are 1 and therefore the correct output is $y = 1$. However, using the aforementioned attack strategy, if the adversarial server succeeds in forcing the reconstructed output to be $y = 0$, it can still potentially generate convincing local proofs. This is because the local proof for C_i only guarantees that there exists b_{1-i} such that the AND of b_i and b_{1-i} is 0.

  The issue with our previous approach based on DP-robust verification is that our simulator can only check validity of the proof share sent by an adversarial server for the "correct" y. Here, since the resulting proof maybe w.r.t. a different $\bar{y}$, DP-robust verification does not suffice. We need a stronger version of DP-robust verification in this setting, that also allows the adversary to maul the statement w.r.t. which the proof is generated.

  Unfortunately, we do not know how to prove this *strong* DP-robust verification for the splittable zk-SNARGs considered in Section 2.2 and Section 2.3. To circumvent this issue, in Section 6, we show how to combine our ideas from the single-client setting with information-theoretic MAC-based ideas from [ADOS22] (see Section 1.3) to obtain a secure multi-client vHSS.

– *Input Sharing:* Unlike in the single-client setting, where the client can use a *secret-key* HSS scheme to compute shares of its input, in the multi-client setting, it is easier for the clients to compute HSS shares of their respective inputs if they start with a public-key HSS scheme. However, this requires the public key and the requisite evaluation keys to be generated in a secure manner. Moreover, given the outcome of this setup procedure, in case some of the clients may also be malicious, we also need to ensure that they send well-formed HSS shares of their inputs. We address this as follows:

1. *Semi-Honest Clients:* When clients are semi-honest, our compiler can be used to transform any public-key HSS scheme with at most negligible correctness error into a multi-client ve-HSS using a suitable Splittable zkSNARG, assuming a secure "public-key setup".

   Abram et al. [ADOS22] demonstrate that for DCR-based [OSY21, RS21] and class group-based [ADOS22] HSS schemes, there exists a maliciously secure one-round distributed protocol for public-key setup that only requires making a black-box use of cryptography. When instantiated with these HSS schemes, our compiler yields a fully black-box solution for multi-client ve-HSS, where the public key setup can be done jointly by the servers using the distributed protocol from [ADOS22].

   Alternatively, we can instantiate our compiler using other HSS schemes. However, since black-box protocols for public-key setup are not known for these other HSS schemes, the setup in these cases must be done by a trusted entity.[13]

2. *Malicious Clients:* Abram et al. [ADOS22] present a black-box, one-round, maliciously secure input sharing protocol (in the random oracle model) for HSS schemes based on DCR [OSY21,RS21] and class groups [ADOS22]. We leverage this (and their distributed protocol for key setup) to demonstrate that our compiler can be instantiated with these HSS schemes to obtain a black-box solution for multi-client ve-HSS.[14]

2.5 Applications

In this section, we discuss applications of our ve-HSS schemes to private delegation of different types of functions.

Delegation of Non-Cryptographic Functions. Our notion of ve-HSS naturally provides a framework for private outsourcing of computation, whether by a single client or a group of clients, to two or more servers. It is easy to see that for non-cryptographic functions, our constructions offer a black-box solution to this problem. As discussed in Section 1.2, this offers several advantages over an FHE based delegation.

Delegation of zkSNARK Computation. We now address the problem of private outsourcing of zk-SNARK computation to a group of untrusted servers by a single client. Specifically, given a public statement x, consider a client who wishes to obtain a proof (using a specific zkSNARK scheme Π) attesting that it knows some witness w such that $C(x, w) = 1$. Instead of computing the proof itself, the client wants to outsource this computation to a group of untrusted servers. This problem has been explored in several recent works [GGJ+23, CLMZ23, LZW+24, GGW24].

Since the computation of a zkSNARK is inherently cryptographic, naïvely using any delegation framework for outsourcing this task may lead to non-black-box use of the cryptographic operations involved in generating the zkSNARK, even if the original delegation scheme is black-box in the underlying cryptographic primitives used in its construction. However, we demonstrate that our specific construction of

[13] This setup could also be implement jointly by the servers using a generic interactive maliciously secure MPC protocol. However, this would result in a non-black-box use of cryptography during the key setup phase.

[14] We note that initializing our compiler with other HSS schemes, for which such black-box protocols for public-key setup and input sharing are not currently known, does not result in a fully black-box solution.

single-client ve-HSS yields a solution for delegating zkSNARK computation that remains black-box in the cryptographic operations used in its generation.

In particular, we observe that if the zkSNARK Π the client wishes to employ is a splittable zkSNARK, then our construction of a single-client ve-HSS provides a fully black-box solution as follows:

1. The client first uses HSS Share to compute shares of $(x_{\mathsf{priv}} = w)$.

2. The client then sends $(x_{\mathsf{pub}} = x)$ along with these HSS shares of x_{priv} to the servers and instructs them to compute C using our single-client ve-HSS (instantiated with Π).

3. Recall that, in our ve-HSS construction, the output of Rec algorithm contains the output $C(x, w) = 1$ a local proof Π_{local} and a public proof Π_{pub}. Here Π_{pub} is a zkSNARK proof for the relation $\mathcal{R}_{\mathsf{pub}} = \{(C, x = x_{\mathsf{pub}}, 1) \mid \exists w = x_{\mathsf{priv}},\ \text{s.t.}, C(x, w) = 1\}$. This corresponds exactly to the zkSNARK that the client wanted to delegate in the first place, so the servers in this construction simply output their respective shares of Π_{pub} and ignore the shares of the local proofs.

This yields a fully non-interactive solution for the problem of private delegation of zkSNARK computation. As discussed in Section 1.2, in contrast, prior MPC-based approaches [GGJ+23, CLMZ23, LZW+24] require a large amount of communication amongst the servers. An advantage of our solution over the FHE based approach proposed in [GGW24] is that the servers can independently compute shares of the delegated zkSNARK proof, without requiring additional intervention from the client.

Delegation of Collaborative zkSNARK Computation. Next, we consider the setting where a group of mutually distrustful clients wish to *jointly* compute a zkSNARK for some statement x using their combined witnesses $(w_i)_{i\in[n]}$ for the relation: $\mathcal{R} : \{(C, x) \mid \exists (w_i)_{i\in[n]}, \text{st.}, C(x, (w_i)_{i\in[n]}) = 1\}$. Such zkSNARKs were recently introduced in [OB22], and are referred to as Collaborative zkSNARKs.

We demonstrate that our construction for multi-client ve-HSS, yields a black-box approach for private delegation of splittable collaborative zkSNARKs. This can essentially be viewed as a multi-client analogue of the previous application. While the main ideas used for enabling this application are similar to those used in the previous application, there are some distinctions:

1. Unlike in the single-prover setting, where the prover knows whether it has a valid witness corresponding to the given statement and relation, in collaborative zkSNARKs, no individual party owns the entire witness. Consequently, the clients do not know if the statement is true or not. As a result, the definition of zero-knowledge is slightly different in collaborative zkSNARKs. Specifically, in this setting, privacy of the honest clients' witnesses must be guaranteed irrespective of whether statement is true or false.

   When using our multi-client ve-HSS scheme for delegating the computation of collaborative zkSNARKs, we propose the following approach to always ensure privacy of the honest clients' witnesses: similar to our previous application, upon receiving x and HSS shares of the witnesses, the servers compute (using the a simplified version of our multi-client ve-HSS) shares of $C(x, (w_i)_{i\in[n]})$ and the public proof (while ignoring the local proofs). In this application, before revealing their shares of the public proof, we instruct the servers to use HSS Eval to multiply the output of $C(x, (w_i)_{i\in[n]})$ with the public proof.

   In case $C(x, (w_i)_{i\in[n]}) = 1$, the resulting value corresponds the the desired collaborative zkSNARK. Privacy of the honest clients' witnesses follows from the zero-knowledge property of the underlying zkSNARK. And in case $C(x, (w_i)_{i\in[n]}) = 0$, above modification ensures that no information (apart from the outcome of $C(x, (w_i)_{i\in[n]})$) about the honest clients' witnesses is revealed.

2. As discussed in Section 2.4, we require clients to perform information-theoretic MAC checks in the ve-HSS construction, based on ideas from [ADOS22], to ensure that adversarial servers cannot tamper with the inputs of clients and deceive them into accepting an incorrect output. Thus, our collaborative

zkSNARK construction requires the servers to also participate in the input-sharing phase to send HSS input shares of their individual MAC keys. The servers then locally perform the MAC check to verify that the relation circuit was evaluated correctly. Since at least one server is assumed to be honest in the collaborative zkSNARK model, the soundness of the MAC check ensures that the statement is indeed verified with the witness shared by clients. Moreover, since the MAC check is run by the servers, the clients are no longer required after the input-sharing phase.

3 Preliminaries

Basic Notation. We use $[x, y]$ to denote the set $\{x, x+1, \ldots, y\}$ where $x, y \in \mathbb{Z}$ and $x \leq y$. We use $\mathbf{x} \in \mathcal{X}^\ell$ to denote the tuple of ℓ elements in $\mathcal{X}$ and $|\mathbf{x}| = \ell$ to denote the length of $\mathbf{x}$. For a function $f : \mathcal{X} \to \mathcal{Y}$ and tuple $\mathbf{x} = (x_1, \ldots, x_\ell) \in \mathcal{X}^\ell$ we use $\mathbf{y} = f(\mathbf{x})$ to denote $\mathbf{y} = (f(x_1), \ldots, f(x_\ell)) \in \mathcal{Y}^\ell$. We use λ as the security parameter, $\mathsf{negl}(\lambda)$ to denote negligible functions, and $\mathsf{poly}(\lambda)$ to denote polynomials in λ. For two probability ensembles $\{A_i\}_i$ and $\{B_i\}_i$, we use $\{A_i\}_i \overset{\text{p}}{\equiv} \{B_i\}_i$ to denote that the ensembles are identical, $\{A_i\}_i \overset{\text{s}}{\approx} \{B_i\}_i$ to denote that the ensembles are statistically close and $\{A_i\}_i \overset{\text{c}}{\approx} \{B_i\}_i$ to denote that the ensembles are computationally indistinguishable. We use $\Pr[E \mid A]$ to denote the probability of an event E in an experiment defined by executing A.

Given a ring R, we will slightly abuse notation by treating a bit $x \in \{0, 1\}$ as an element of R via the natural mapping: 0 maps to the additive identity 0_R and 1 maps to the multiplicative identity 1_R. For a circuit C over a ring R, the depth of C is the maximum number of gates on any path from an input to an output, while the multiplicative depth of C is the maximum number of multiplication gates on any such path. In this work, we only consider circuits with 2-input gates.

Additive Secret Sharing. Let $\mathbb{G}$ be a group and let $m \in \mathbb{N}$. We use $[x] = ([x]_1, \ldots, [x]_m) \in \mathbb{G}^m$ to denote an m-party additive secret sharing of a secret $x \in \mathbb{G}$, where the i-th party holds the share $[x]_i$ and $\sum_{i=1}^m [x]_i = x$. Observe that the scheme is *additively homomorphic* i.e., it allows parties to compute on the secrets by carrying out the same operations on corresponding shares. For a subset $\mathcal{I} \subset [1, m]$, we use $[x]_\mathcal{I}$ to denote $\{[x]_i\}_{i \in \mathcal{I}}$. For a tuple $\mathbf{x} = (x_1, \ldots, x_\ell) \in \mathbb{G}^\ell$, we use $[\mathbf{x}]$ to denote the sharings $([x_1], \ldots, [x_\ell])$ and use $[\mathbf{x}]_i$ to denote the shares $([x_1]_i, \ldots, [x_\ell]_i)$ of the i-th party. We say $[x] \in R^m$ is an additive sharing of $x \in R$, for a ring R, to denote that it is an additive sharing over the group $(R, +)$. In this case, we say $[b]$ is an additive sharing of a bit $b \in \{0, 1\}$ over R if $[b]$ reconstructs to the additive identity 0_R when $b = 0$ and it reconstructs to the multiplicative identity 1_R when $b = 1$.

3.1 Homomorphic Secret Sharing

An HSS scheme allows a client to secret share its inputs among a set of servers, which can then *non-interactively* evaluate a function on the shared inputs to obtain an additive sharing of the output. Informally, an HSS scheme can be viewed as a distributed variant of homomorphic encryption where inputs are secret shared instead of being encrypted and parties obtain additive shares of the output upon evaluation instead of an encrypted output. The inputs need not be secret shared using an additive secret sharing scheme and we use $\langle \mathbf{x} \rangle = (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m)$ to denote the HSS input shares for a tuple $\mathbf{x}$ where the i-th server's share is denoted by $\langle \mathbf{x} \rangle_i$. For any $\mathcal{I} \subseteq [1, m]$, we use $\langle \mathbf{x} \rangle_\mathcal{I}$ as a shorthand to denote $\{\langle \mathbf{x} \rangle_i\}_{i \in \mathcal{I}}$.

We first recall the definition of HSS in the single client and multiple server setting and then discuss public-key HSS which extends the definition to multiple clients.

Definition 1 (Homomorphic Secret Sharing [BGI16a])**.** *Let $m := m(\lambda)$ be an integer-valued polynomial, R be a ring and let $\mathbb{C}_{\mathsf{hss}}$ be a class of circuits over R. An m-server Homomorphic Secret Sharing (HSS) scheme for $\mathbb{C}_{\mathsf{hss}}$ is a pair of algorithms $\mathsf{HSS} = (\mathsf{Share}, \mathsf{Eval})$ with the following syntax.*

- $\mathsf{Share}(1^\lambda, \mathbf{x}) \to (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m)$ *is a* PPT *algorithm that takes as input* $\mathbf{x} \in R^*$ *and for each* $i \in [1, m]$, *outputs a share* $\langle \mathbf{x} \rangle_i$ *for the* i*-th server.*

- $\mathsf{Eval}(i, \mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i) =: [\mathbf{y}]_i$ *is a polynomial time algorithm that takes as input a server index* i*, a circuit* $\mathsf{C} \in \mathbb{C}_{\mathsf{hss}}$*, common server input* $\mathbf{x}_{\mathsf{pub}} \in R^*$*, and the share of the* i*-th server* $\langle \mathbf{x} \rangle_i$ *and computes an additive share of the circuit output* $[\mathbf{y}]_i$ *over* R.

An HSS scheme should satisfy the following properties.

- ***Correctness:*** *For all polynomials* $\mathsf{poly}(\cdot)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, circuits* $\mathsf{C} \in \mathbb{C}_{\mathsf{hss}}$*, and inputs* $\mathbf{x}_{\mathsf{pub}} \in R^*$ *and* $\mathbf{x} \in R^*$ *we have*

$$\Pr\left[[\mathbf{y}]_1 + \ldots + [\mathbf{y}]_m \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \;\middle|\; \begin{array}{c} (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m) \leftarrow \mathsf{Share}(1^\lambda, \mathbf{x}) \\ [\mathbf{y}]_i := \mathsf{Eval}(i, P, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i),\ \forall i \in [1, m] \end{array} \right] \leq \mathsf{negl}(\lambda)$$

where $|P|, |\mathbf{x}_{\mathsf{pub}}|, |\mathbf{x}| \leq \mathsf{poly}(\lambda)$*. Alternatively, in a* δ*-correct scheme, the above probability is at most* $\delta + \mathsf{negl}(\lambda)$ *and the* Eval *algorithm is allowed to run in time polynomial in* $1/\delta$.

- ***Security:*** *For every polynomial size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *we have*

$$\Pr\left[b' = b \;\middle|\; \begin{array}{r} (\mathsf{st}, \mathcal{I}, \mathbf{x}_0, \mathbf{x}_1) \leftarrow \mathcal{A}(1^\lambda) \\ b \leftarrow \{0,1\} \\ (\langle \mathbf{x}_b \rangle_1, \ldots, \langle \mathbf{x}_b \rangle_m) \leftarrow \mathsf{Share}(1^\lambda, \mathbf{x}_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \langle \mathbf{x}_b \rangle_{\mathcal{I}}) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{I} \subset [1, m]$*,* $\mathbf{x}_0, \mathbf{x}_1 \in R^*$*, and* $|\mathbf{x}_0| = |\mathbf{x}_1|$.

Observe that the output of Eval is *compact* since it computes an additive share i.e., the size of the output share depends only on the size of the circuit output and is independent of the size of the inputs and the circuit. Moreover, since Share is PPT, the size of each input share is polynomial in the size of the circuit input $\mathbf{x}$ and the security parameter but independent of the circuit size.

Public-key HSS. A public-key HSS scheme [BGI16a] allows computation on the combined inputs of multiple clients. It includes an additional key generation algorithm that outputs a public key pk and an evaluation key ek_i for each server. Any client can encrypt its input x using the public key pk to compute the ciphertext $\widetilde{x}$. Each server can then use its evaluation key and the ciphertexts provided by the clients to locally evaluate a circuit on the joint inputs of all clients, and obtain an additive share of the circuit output.

Definition 2 (Public-key Homomorphic Secret Sharing [BGI16a])**.** *Let* $m := m(\lambda)$ *be an integer-valued polynomial,* R *be a ring and let* $\mathbb{C}_{\mathsf{hss}}$ *be a class of circuit over* R*. An* m*-server public-key Homomorphic Secret Sharing (HSS) scheme for* $\mathbb{C}_{\mathsf{hss}}$ *is a tuple of algorithms* $\mathsf{HSS} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval})$ *with the following syntax.*

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m)$ *is a* PPT *algorithm that outputs a public key* pk *and* m *evaluation keys, where* ek_i *is the evaluation key of the* i*-th server.*

- $\mathsf{Enc}(\mathsf{pk}, x) \to \widetilde{x}$ *is a* PPT *algorithm that takes a value* $x \in R$ *and outputs a ciphertext* $\widetilde{x}$.

- $\mathsf{Eval}(\mathsf{ek}_i, P, \mathbf{x}_{\mathsf{pub}}, \widetilde{x}_1, \ldots, \widetilde{x}_n) =: [\mathbf{y}]_i$ *is a polynomial time algorithm that takes as input the evaluation key* ek_i *of the* i*-th server, a circuit* $\mathsf{C} \in \mathbb{C}_{\mathsf{hss}}$*, common server input* $\mathbf{x}_{\mathsf{pub}} \in R^*$*, and* n *ciphertexts corresponding to the client inputs and computes an additive share of the circuit output* $[\mathbf{y}]_i$ *over* R.

A public-key HSS scheme should satisfy the following properties.

– ***Correctness:*** *For all polynomials* $\mathsf{poly}(\cdot)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda, n \in \mathbb{N}$*, circuits* $\mathsf{C} \in \mathbb{C}_{\mathsf{hss}}$*, and inputs* $\mathbf{x}_{\mathsf{pub}} \in R^*$ *and* $x_1, \ldots, x_n \in R$ *we have*

$$\Pr\left[\begin{array}{c} [\mathsf{y}]_1 + \ldots + [\mathsf{y}]_m \\ \neq \\ \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, x_1, \ldots, x_n) \end{array} \middle| \begin{array}{r} (\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \widetilde{x}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i),\ \forall i \in [1,n] \\ [\mathsf{y}]_i \coloneqq \mathsf{Eval}(\mathsf{ek}_i, P, \mathbf{x}_{\mathsf{pub}}, \widetilde{x}_1, \ldots, \widetilde{x}_n),\ \forall i \in [1,m] \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $|P|, n, |\mathbf{x}_{\mathsf{pub}}| \leq \mathsf{poly}(\lambda)$*.*

– ***Security:*** *For every polynomial size adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $\mathcal{I} \subset [1,m]$ *we have*

$$\Pr\left[\begin{array}{c} b' = b \end{array} \middle| \begin{array}{r} (\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}\big(1^\lambda, \mathsf{pk}, \{\mathsf{ek}_i\}_{i \in \mathcal{I}}\big) \\ b \leftarrow \{0,1\} \\ \widetilde{x}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \widetilde{x}_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $x_0, x_1 \in R$*.*

As in the case of single client HSS, the fact that Eval computes additive shares of the circuit output implies that its output is *compact* — the size of the output share depends only on the size of the circuit output and is independent of the size of the input and circuit.

Remark 1 (Simulating Output Shares in HSS)**.** *Observe that the security of HSS schemes only ensure that the input shares (or ciphertexts in case of public-key HSS) sent to corrupt servers can be simulated. On the other hand, applications of HSS will often involve reconstructing the output, which will in turn require simulating the output shares of the honest server in the security proof. However, the security and correctness properties of HSS allow simulating the honest server's output share too. Specifically, we first simulate input shares or ciphertexts sent by clients to corrupt servers and then compute the shares of the output* $[\mathsf{y}]_{\mathcal{I}}$ *obtained by corrupt servers upon running* Eval*. The output share of the honest server can then be computed using the circuit output* y *and the additive shares* $[\mathsf{y}]_{\mathcal{I}}$ *obtained by corrupt servers. Crucially, since evaluation is deterministic, the corrupt servers indeed obtain* $[\mathsf{y}]_{\mathcal{I}}$ *upon HSS evaluation. The correctness property of HSS then ensures that the simulated output share is indistinguishable from the output share of the honest server.*

Restricted Multiplication Straight-line Programs. A large class of existing HSS schemes [BGI16a, BKS19, OSY21, RS21, ADOS22] only support evaluating Restricted Multiplication Straight-line (RMS) programs [Cle90, BGI16a]. An RMS program is a circuit over integers with the restriction that every multiplication is between an input value and an intermediate value of the computation, called a memory value. Boyle et. al [BGI16a] showed that the class of polynomial size RMS programs includes the class of polynomial size branching programs, which is in turn known to contain the class of NC^1 circuits.

Definition 3 (Restricted Multiplication Straight-line Program [Cle90, BGI16a])**.** *A Restricted Multiplication Straight-line (RMS) Program* P *consists of a magnitude bound* $B_{\mathsf{rms}} \in \mathbb{N}$ *and an arbitrary sequence of the following four instructions.*

– $\mathsf{LoadInput}(\mathsf{I}_x) \rightarrow \mathsf{M}_x$*: Load the value of the input* I_x *to the memory* M_x*.*

– $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y) \rightarrow \mathsf{M}_z$*: Add the values of the memory* M_x *and* M_y *and assign the result to the memory* M_z*.*

– $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y) \rightarrow \mathsf{M}_z$*: Multiply the value of the input* I_x *by the value of the memory* M_y *and assign the result to the memory* M_z*.*

– $\mathsf{Output}(\mathsf{M}_z, p_{\mathsf{rms}}) \to z \bmod p_{\mathsf{rms}}$: *Output the value of the memory* M_z *reducing it modulo* p_{rms}.

If at any step of the execution, the magnitude of a memory value exceeds the bound B_{rms}*, the output of the program on the corresponding input is defined to be* $\perp$*. The size of an RMS program is defined as the number of instructions.*

In an HSS scheme for RMS programs with bound B_{rms}, servers obtain an additive sharing of the program output upon HSS evaluation when the RMS program does not evaluate to $\perp$; otherwise, they obtain $\perp$. In particular, computation takes place over the ring $\mathbb{Z}_{p_{\mathsf{rms}}}$.

The following lemma states that any polynomial-size log-depth circuit over an integer ring can be computed by a polynomial-size RMS program.

Lemma 5. *For any n-input polynomial size circuit* C *over* $\mathbb{Z}_p$ *with* $O(\log n)$*-depth, there exists a polynomial size RMS program* P *with bound* $B_{\mathsf{rms}} \in p^{O(n)}$ *such that for all* $\mathbf{x} \in \mathbb{Z}_p^n$, $P(\mathbf{x}) = \mathsf{C}(\mathbf{x})$.

Proof. P takes the same inputs as C and computes a memory value M_y for each intermediate value y of the circuit, in the order determined by a breadth-first traversal of C. Specifically, for each input x, P computes a $\mathsf{LoadInput}(\mathsf{I}_x)$ instruction and obtains a memory value M_x. For every gate g of C at depth i, if g is an addition gate with inputs y and z, P computes the $\mathsf{Add}(\mathsf{M}_y, \mathsf{M}_z)$ instruction. On the other hand, if g is a multiplication gate with inputs y and z, P first computes $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$ for each program input x and then recomputes the RMS program used to compute M_x, except that it replaces the memory value $\mathsf{M}_{z'}$ with $\mathsf{M}_{z' \cdot y}$ for each instruction in this sub-program. Finally, it computes $\mathsf{Output}(\mathsf{M}_z, p)$ for the output wire z of the circuit.

Observe that $\mathsf{C}(\mathbf{x})$ when computed over the integers is less than p^n for all $\mathbf{x} \in \mathbb{Z}_p^n$ since C has depth $O(\log n)$. Since P has bound $p^{O(n)}$, it follows from our description of P that for every input $\mathbf{x} \in \mathbb{Z}_p^n$, $P(\mathbf{x}) = \mathsf{C}(\mathbf{x})$. Next, note that computing the output of a gate at depth i of C from the inputs $\mathbf{x}$ requires at most twice the number of instructions than computing the output of a gate at depth-$(i-1)$. Since C has polynomial size and $O(\log n)$-depth, it then follows from a straightforward counting argument that the size of P is polynomial in n. ■

3.1.1 HSS Schemes from Prior Works

We briefly summarize the HSS schemes from prior works. The first HSS scheme was proposed by Boyle et al. [BGI16a] where they constructed a 2-server HSS scheme and a 2-server public-key HSS scheme for evaluating polynomial size RMS programs, assuming the hardness of the Decisional Diffie-Hellman (DDH) assumption. However, their HSS schemes suffer from inverse polynomial correctness error and only supports evaluating RMS programs with a magnitude bound that is polynomial in the security parameter. Subsequently, a number of works have proposed 2-server public-key HSS schemes with negligible correctness error [BKS19, OSY21, RS21, ADOS22]. These schemes support evaluating polynomial size RMS programs, where the magnitude bound is exponential in the security parameter.

In more detail, [BKS19] construct HSS schemes from the Learning With Errors (LWE) assumption while [OSY21, RS21] construct HSS schemes based on the Decisional Composite Residuosity (DCR) assumption. The primary difference between [OSY21] and [RS21] is that the HSS schemes of [OSY21] are based on the Paillier and Brakerski-Goldwasser [BG10] encryption schemes while the HSS scheme of [RS21] is based on the Damgård-Jurik encryption scheme [DJ01]. Abram et al. [ADOS22] generalize and extend the constructions of [OSY21, RS21] by presenting a general group-theoretic framework which they call as the Non-Interactive Discrete Log Sharing (NIDLS) framework. They then show that their framework can be instantiated with class groups [CL15]. Since any public-key HSS scheme is also a single client HSS scheme, these works also imply 2-server HSS schemes from the same assumptions.

The HSS scheme of [DIJL23], based on sparse LPN, supports more than 2 servers but has inverse polynomial correctness error and only allows evaluating multivariate polynomials with polynomially many monomials and of total degree at most $O(\log \lambda / \log\log \lambda)$. HSS schemes for evaluating larger classes of circuits rely on strong assumptions [BGI15, DHRW16, BGI+18, COS+22]. These schemes support evaluating any polynomial size circuit as well as support having more than 2 servers. Specifically, for all positive integers m, m-server HSS schemes for P/poly are known either assuming the existence of sub-exponentially secure indistinguishability obfuscation and one-way functions [BGI15] or from circular secure Additive-Function Sharing (AFS) spooky encryption [DHRW16, BGI+18, COS+22]. AFS spooky encryption is known from LWE [DHRW16, COS+22]. These HSS schemes [BGI15, COS+22] also support evaluating polynomial size circuits over integer rings that are exponentially large in the security parameter. The m-server HSS schemes [DHRW16, COS+22] from AFS-spooky encryption are also public-key HSS schemes.

3.2 Succinct Non-interactive Arguments

A Succinct Non-interactive Argument (SNARG) allows a prover holding a witness w to compute a proof π that convinces a verifier about the validity of a given statement x, such that the size of π and the verification time are sublinear in the size of w. A zero-knowledge SNARG (zkSNARG) additionally ensures that the proof does not leak any additional information about the witness w. In this work, we consider zkSNARGs for the circuit satisfiability language.

Circuit Satisfiability Language. The circuit satisfiability language SAT is defined as

$$\mathsf{SAT} = \{(\mathsf{C}, x, y) \mid \exists w,\ \mathsf{C}(x, w) = y\}$$

where $\mathsf{C} : R^{\ell_{\mathsf{inp}}} \times R^{\ell_{\mathsf{wit}}} \to R^{\ell_{\mathsf{out}}}$ is a circuit, R is a ring, ℓ_{inp}, ℓ_{wit} and ℓ_{out} are natural numbers, $x \| y \in R^{\ell_{\mathsf{stmt}}}$ is the statement and $\ell_{\mathsf{stmt}} = \ell_{\mathsf{inp}} + \ell_{\mathsf{out}}$. The description of the circuit also defines ℓ_{inp}, ℓ_{out}, ℓ_{wit}, and R. We will overload notation and use SAT to also refer to the circuit satisfiability relation so that $(\mathsf{C}, x, y; w) \in \mathsf{SAT}$ when $\mathsf{C}(x, w) = y$.

Definition 4 (Zero-knowledge Succinct Non-interactive Argument)**.** *Let $\mathbb{C}_{\mathsf{ZK}}$ be a class of circuits. A zero-knowledge Succinct Non-interactive Argument (zkSNARG) for $\mathbb{C}_{\mathsf{ZK}}$-circuit satisfiability is a triple of algorithms* $\mathsf{SNARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *with the following syntax.*

- $\mathsf{Setup}(1^\lambda, \mathsf{C}) \to \mathsf{crs}$ *is a* PPT *algorithm that takes the description of a circuit $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ as input and outputs a common reference string* crs.
- $\mathsf{Prove}(\mathsf{crs}, x, y, w) \to \pi$ *is a* PPT *algorithm that takes the common reference string* crs*, a statement (x, y) and a witness w as input and outputs a proof π.*
- $\mathsf{Verify}(\mathsf{crs}, x, y, \pi) \to b$ *is a* PPT *algorithm that takes the common reference string* crs*, a statement (x, y) and a proof π as input and outputs a bit b indicating if the proof is accepted or rejected.*

We require a zkSNARG to satisfy the following properties.

- ***Succinctness:*** *There exists polynomial $p(\cdot)$ and a sublinear function $g(x) \in o(x)$ such that for all polynomials* $\mathsf{poly}(\cdot)$ *and $\lambda, \ell_{\mathsf{stmt}}, \ell_{\mathsf{wit}} \in \mathbb{N}$ where $\ell_{\mathsf{stmt}}, \ell_{\mathsf{wit}} \leq \mathsf{poly}(\lambda)$, the size of the proof π output by* Prove *for a statement of length ℓ_{stmt} and witness of length ℓ_{wit} is at most $p(\lambda) g(\ell_{\mathsf{wit}})$ and the runtime of* Verify *is at most $p(\lambda \ell_{\mathsf{stmt}}) g(\ell_{\mathsf{wit}})$.*
- ***Completeness:*** *For all polynomial size adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ we have*

$$\Pr\left[\begin{array}{c|c} \mathsf{C}(x, w) = y & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ \wedge & (x, y, w) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \mathsf{Verify}(\mathsf{crs}, x, y, \pi) = 0 & \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, y, w) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

– ***Soundness:*** *For all polynomial size adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ *we have*

$$\Pr\left[\begin{array}{c}(\mathsf{C}, x, y) \notin \mathsf{SAT} \\ \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \pi) = 1\end{array} \;\middle|\; \begin{array}{c}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, y, \pi) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

A zkSNARG is said to have non-adaptive *soundness if for all polynomial size adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ *we have*

$$\Pr\left[\begin{array}{c}(\mathsf{C}, x, y) \notin \mathsf{SAT} \\ \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \pi) = 1\end{array} \;\middle|\; \begin{array}{c}(x, y, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ \pi \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{crs})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

– ***Zero-Knowledge:*** *There exists a* PPT *simulator* Sim *such that for all polynomial size adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *so that for all* $\lambda \in \mathbb{N}$ *and circuit* $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ *we have*

$$\left|\Pr\left[\mathcal{A}(\mathsf{st}, \pi) = 1 \;\middle|\; \begin{array}{r}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, w, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ y := \mathsf{C}(x, w) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, y, w)\end{array}\right] - \Pr\left[\mathcal{A}(\mathsf{st}, \pi) = 1 \;\middle|\; \begin{array}{r}(\mathsf{crs}, \mathsf{st}_s) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{C}) \\ (x, w, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ y := \mathsf{C}(x, w) \\ \pi \leftarrow \mathsf{Sim}(\mathsf{st}_s, x, y)\end{array}\right]\right| \leq \mathsf{negl}(\lambda).$$

Succinct Non-interactive Arguments of Knowledge. A zero-knowledge Succinct Non-interactive Argument of Knowledge (zkSNARK) is a zkSNARG that additionally has the knowledge soundness property. Knowledge soundness requires that for all polynomial size adversaries $\mathcal{A}$ there exists a polynomial size extractor $\mathsf{Ext}_{\mathcal{A}}$ and a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ we have

$$\Pr\left[\begin{array}{c}\mathsf{C}(x, w) \neq y \\ \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \pi) = 1\end{array} \;\middle|\; \begin{array}{r}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, y, \pi) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ w \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Batch Arguments. Non-interactive batch arguments (BARGs) allow constructing and verifying proofs for multiple instances of an NP language. In this work, we consider BARGs for boolean circuit satisfiability. Let $\mathsf{C} : \{0,1\}^{\ell_{\mathsf{stmt}}} \times \{0,1\}^{\ell_{\mathsf{wit}}} \rightarrow \{0,1\}$ be a boolean circuit consisting of s NAND gates. For a positive integer $n \in \mathbb{N}$, the batch boolean circuit satisfiability language BSAT_n is defined as

$$\mathsf{BSAT}_n = \left\{(\mathsf{C}, \mathbf{x}_1, \ldots, \mathbf{x}_n) \mid \forall i \in [1, n]\ \exists \mathbf{w}_i \in \{0,1\}^{\ell_{\mathsf{wit}}}, \mathsf{C}(\mathbf{x}_i, \mathbf{w}_i) = 1\right\}.$$

A BARG is required to satisfy the succinctness, completeness and non-adaptive soundness properties, as defined in Definition 4, except that the statement now consists of n circuit inputs as defined in BSAT_n. A BARG that is additionally zero-knowledge is called a zkBARG. Observe that it suffices for a BARG to amortize the cost of verification across multiple instances to achieve succinctness. Specifically, a BARG is succinct if the proof size and verification times are sublinear in $n\ell_{\mathsf{wit}}$ but linear in $n + \ell_{\mathsf{wit}}$. We will also view BSAT_n as the circuit satisfiability language for SIMD boolean circuits. Specifically, an instance $(\mathsf{C}, \mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathsf{BSAT}_n$ can alternatively be viewed as an instance of a circuit satisfiability language for a class of SIMD boolean circuits C' such that C' consists of n parallel copies of C and is satisfiable iff each copy of C with input $\mathbf{x}_i$ is satisfiable. Thus, a BARG is essentially a SNARG for SIMD-circuit satisfiability with non-adaptive soundness.

4 Splittable zkSNARG

In this section, we introduce the notion of Splittable zkSNARGs, which are used to capture the properties of zkSNARGs required by our ve-HSS construction. We begin by defining Splittable zkSNARGs, then establish the key properties of these zkSNARGs when proofs are generated by distributed provers, and finally present two instantiations of Splittable zkSNARGs.

The main property of a Splittable zkSNARG is a structural requirement called the *split prover* property, which allows the prover to be split into a non-linear phase followed by a linear phase, where the non-linear phase is "simple" to compute. In our ve-HSS construction, the non-linear phase will be computed using an HSS scheme and the linear phase will be computed over the additive shares obtained from HSS evaluation, leveraging the linearity of additive secret sharing. Since most HSS schemes only support evaluating a restricted class of circuits, we parameterize Splittable zkSNARGs by the class of circuits $\mathbb{C}_{\mathsf{NL}}$ required for computing the non-linear phase of the zkSNARG, when proving satisfiability of circuits in a set $\mathbb{C}_{\mathsf{ZK}}$. Our ve-HSS scheme can then be realized using an HSS scheme that supports evaluating circuits in $\mathbb{C}_{\mathsf{NL}}$.

In addition to the split prover property, a Splittable zkSNARG also satisfies the *succinct randomness* and *robust verification* properties. Succinct randomness requires that the length of the random tape required to compute the proof is sublinear in the size of the circuit being proved satisfiable. This ensures that the size of the input shares in the ve-HSS construction is sublinear in the circuit size. Robust verification is a security property that requires that any adversarially introduced additive error to a valid proof will cause the proof to fail verification. In Section 4.1, we show that the robust verification property naturally lends itself to security in a distributed prover setting. Looking ahead, this is crucial to the security of our ve-HSS construction, ensuring that an adversary does not learn any additional information about the private witness, even if they cause the proof to be computed incorrectly.

Definition 5 (Splittable zkSNARG)**.** *Let $\mathbb{C}_{\mathsf{NL}}$ and $\mathbb{C}_{\mathsf{ZK}}$ be two classes of circuits. A zkSNARG* $\mathsf{SNARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is said to be a $\mathbb{C}_{\mathsf{NL}}$-simple Splittable zkSNARG for $\mathbb{C}_{\mathsf{ZK}}$-circuit satisfiability if it additionally satisfies the following properties.*

- ***Split Prover:*** *There exist* PPT *algorithm* $\mathsf{Prove}_{\mathsf{NL}}$ *and polynomial time algorithm* $\mathsf{Prove}_{\mathsf{L}}$ *such that for any circuit $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$, we have the following.*
  - $\mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, x, y, w) \to M_{\mathsf{NL}}$ *takes the common reference string* crs*, circuit* C*, statement (x, y) and witness w as input and outputs a tuple M_{NL}.*
  - $\mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{NL}}) =: \pi$ *takes M_{NL} as input and outputs a proof π. The map $\mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, \cdot)$ is a group homomorphism.*

  We require that for all $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$

$$\left\{ \pi \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, y, w) \end{array} \right\}_{\lambda, x, w} \overset{c}{\approx} \left\{ \pi \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ M_{\mathsf{NL}} \leftarrow \mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, x, y, w) \\ \pi := \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{NL}}) \end{array} \right\}_{\lambda, x, w}$$

  where $\lambda \in \mathbb{N}$ and $y = \mathsf{C}(x, w)$.

- ***Robust Verification:*** *For all polynomial size adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$, we have*

$$\Pr\left[\begin{array}{c} \mathsf{C}(x, w) = y \wedge \\ \epsilon \neq 0 \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \overline{\pi}) = 1 \end{array} \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, y, w, \epsilon) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, y, w) \\ \overline{\pi} := \pi + \epsilon \end{array} \right] \leq \mathsf{negl}(\lambda).$$

– ***Efficiency:*** *A $\mathbb{C}_{\mathsf{NL}}$-simple Splittable zkSNARG should satisfy the following efficiency properties.*

  – $\mathbb{C}_{\mathsf{NL}}$***-simple:*** *For every circuit $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ there exists an efficiently constructible circuit $\mathsf{C}_{\mathsf{NL}} \in \mathbb{C}_{\mathsf{NL}}$ such that for all inputs x and w, and randomness r we have $\mathsf{C}_{\mathsf{NL}}(x, w, r) = \mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, x, y, w; r)$ where $y = \mathsf{C}(x, w)$.*

  – ***Succinct Randomness:*** *There exists polynomial p and a sublinear function $g(x) \in o(x)$ such that for all $\lambda \in \mathbb{N}$ and all ℓ_{inp}-input circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$, the length of the random tape required by $\mathsf{Prove}_{\mathsf{NL}}$ is at most $p(\lambda \ell_{\mathsf{inp}})g(|\mathsf{C}|)$.*

We note that $\mathbb{C}_{\mathsf{NL}}$ is parameterized by the CRS though we omit it from the notation for simplicity. If the Splittable zkSNARG additionally satisfies knowledge soundness, we say it is a Splittable zkSNARK. We will say a zkSNARG is a Splittable zkSNARG, and omit the description of $\mathbb{C}_{\mathsf{ZK}}$ and $\mathbb{C}_{\mathsf{NL}}$, when only referring to the split prover and succinct randomness property of the zkSNARG.

Splittable zkSNARGs from Linear PCPs. It is easy to see that zkSNARGs based on linear PCPs [BCI+13] satisfy the structural properties of a Splittable zkSNARG. In more detail, a linear PCP is an oracle where responses to verification queries are computed as inner products between a PCP string and the query. A zkSNARG can be obtained from a linear PCP by embedding the verifier's queries in the common reference string (CRS) using a linearly homomorphic encoding scheme. The zkSNARG prover holding the witness can first compute the PCP string and then leverage the linear homomorphism of the encoding scheme to compute the inner product of the verification queries embedded in the CRS with the PCP string. These encodings of the inner product serve as the zkSNARG proof. The encoding scheme also ensures privacy of the verification queries embedded in the CRS, which in turn ensures soundness of the zkSNARG. Thus, the prover of a zkSNARG scheme based on linear PCPs can be naturally split into a linear and non-linear phase — the computation of the linear PCP constitutes the non-linear phase and using the CRS to compute the responses to the embedded verification queries constitutes the linear phase.

We also note that zkSNARGs based on linear PCPs can be easily modified to realize the robust verification property.[15] In particular, the setup additionally samples a MAC key K uniformly at random, and the non-linear phase computes the linear PCP M_{NL} as well as a MAC on the linear PCP $K \cdot M_{\mathsf{NL}}$. The linear homomorphism of the encoding scheme allows evaluating the linear phase on M_{NL} and $K \cdot M_{\mathsf{NL}}$ to compute a proof π and a MAC on the proof $K \cdot \pi$. The modified proof π' consists of $\pi' = (\pi, K \cdot \pi)$. The verification algorithm, given the MAC key K and a proof $\pi' = (\pi_1, \pi_2)$, first verifies the MAC on the proof by checking if $K \cdot \pi_1 = \pi_3$ and then runs the zkSNARG verifier on π_1. It is easy to see that the resulting proof preserves zero-knowledge, soundness and the split-prover property. Moreover, it has robust verification because any additive error to the proof will be detected with probability at least $1/2$ by the MAC-check; which can in turn be amplified to detect error with overwhelming probability by computing λ independent MACs. The modified proof system, however, is *designated-verifier*, since it requires the verifier to store the MAC keys generated during setup. Nevertheless, such a proof system remains compatible with our ve-HSS construction, up to some syntactic modifications—specifically, the sharing algorithm will output both the server shares and the private client state.

In Sections 4.2 and 4.3, we discuss two Splittable zkSNARGs, where the split prover property of the zkSNARGs follow from the fact they are based on linear PCPs. Moreover, we show that these zkSNARGs already satisfy the robust verification property, without any modifications. In particular, these zkSNARGs are publicly verifiable and have robust verification, unlike the generic transformation described previously. The non-linear phases of these zkSNARGs can be computed by a large number of existing HSS schemes, and since it does not make use of any cryptographic primitives, computing it within HSS does not make non-black box use of cryptography.

[15] Suggested by an anonymous TCC 2024 reviewer.

4.1 Distributed Prover Setting

As discussed earlier, our ve-HSS construction separates the work of a Splittable zkSNARG prover by running the non-linear phase within an HSS scheme and computing the linear phase over additive shares obtained from HSS evaluation. In this section, we show that Splittable zkSNARGs, when computed in a distributed manner, naturally provide completeness and security properties.

In more detail, we consider a set of distributed provers who are given an additive sharing of the output of the non-linear phase M_{NL} of the Splittable zkSNARG. We aim to show that the provers can non-interactively compute additive shares of a valid proof and that if any subset of the provers behave maliciously, the reconstructed proof will fail to verify. Since the non-linear phase is run within HSS, the security of the ve-HSS construction will follow from the security of the HSS scheme and the security of the Splittable zkSNARG in the distributed prover setting.

DP-Completeness. Distributed prover completeness guarantees that the provers, given additive shares of M_{NL}, can non-interactively compute additive shares of a valid zkSNARG proof. In the definition, the adversary is allowed to choose the shares of M_{NL} for corrupt provers. Intuitively, this means that the reconstructed proof must verify successfully irrespective of how corrupt provers obtain their shares of M_{NL}. In case of our ve-HSS construction, these shares will be computed using an HSS scheme. Without loss of generality, we consider adversaries that corrupt all but one prover.

Lemma 6 (DP-Completeness)**.** *Let* SNARG *be a Splittable zkSNARG for $\mathbb{C}_{\mathsf{ZK}}$-circuit satisfiability. Then for all polynomial size adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda, m \in \mathbb{N}$, where $m > 1$, and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ we have*

$$\Pr\left[\begin{array}{c} \mathsf{C}(x,w) = y \\ \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \pi) = 0 \end{array} \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, y, w, \mathcal{I}, [M_{\mathsf{NL}}]_{\mathcal{I}}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ M_{\mathsf{NL}} \leftarrow \mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, x, y, w) \\ [M_{\mathsf{NL}}]_{i^*} := M_{\mathsf{NL}} - \sum_{i \in \mathcal{I}} [M_{\mathsf{NL}}]_i \\ [\pi]_i := \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, [M_{\mathsf{NL}}]_i),\ \forall i \in [1,m] \\ \pi := \sum_{i=1}^{m} [\pi]_i \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{I} \subset [1,m]$ is a set of $m-1$ corrupt provers and $i^* \notin \mathcal{I}$ is the index of the honest prover.

Proof. Observe that $[M_{\mathsf{NL}}] = ([M_{\mathsf{NL}}]_1, \ldots, [M_{\mathsf{NL}}]_m)$ is an additive sharing of M_{NL}, the output of the non-linear phase of the zkSNARG. Since $\mathsf{Prove}_{\mathsf{L}}$ is a group homomorphism, $[\pi] = ([\pi]_1, \ldots, [\pi]_m)$ is an additive sharing of the proof $\pi := \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{NL}})$. The split prover and completeness properties of SNARG then imply that verification of π fails with at most negligible probability when $\mathsf{C}(x,w) = y$. ■

DP-Robust verification. Distributed prover robust verification guarantees that a proof reconstructed using incorrect shares sent by corrupt provers will fail to verify. Specifically, since the linear phase of the Splittable zkSNARG is deterministic, the proof π is completely determined by the CRS and the output M_{NL} of the non-linear phase. DP-Robust verification then guarantees that if corrupt provers behave maliciously and output incorrect shares of the proof, such that the reconstructed proof $\overline{\pi}$ is not equal to the proof π, then $\overline{\pi}$ will fail to verify with overwhelming probability. We will show that robust verification in the distributed prover setting is implied by the robust verification property of Splittable zkSNARGs. As in the case of DP-completeness, we allow the adversary to choose the corrupt provers' shares of M_{NL} and we consider adversaries that corrupt all but one prover without loss of generality.

This property helps provide a strong privacy guarantee when the proof for a statement consisting of private data is computed in a distributed manner. In particular, DP-robust verification ensures that corrupt provers cannot compromise the privacy of the statement by maliciously computing their proof shares

and learning if the reconstructed proof verifies. This is because any error in the reconstructed proof is introduced only by corrupt parties and verification is guaranteed to fail with overwhelming probability in case of non-zero error—this implies that corrupt parties could have themselves determined if the reconstructed proof verifies successfully. Looking ahead, this is crucial to proving the privacy of our ve-HSS construction.

Lemma 7 (DP-Robust Verification)**.** *Let* SNARG *be a Splittable zkSNARG for $\mathbb{C}_{\mathsf{ZK}}$-circuit satisfiability and let $m \in \mathbb{N}$. Then for all polynomial size adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda, m \in \mathbb{N}$, where $m > 1$, and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ we have*

$$\Pr\left[\begin{array}{c} \mathsf{C}(x,w) = y \\ \wedge \\ \epsilon \neq 0 \\ \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \overline{\pi}) = 1 \end{array} \middle| \begin{array}{r} (x, y, w, \mathcal{I}, [M_{\mathsf{NL}}]_{\mathcal{I}}, [\overline{\pi}]_{\mathcal{I}}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ [M_{\mathsf{NL}}]_{i^*} \coloneqq M_{\mathsf{NL}} - \sum_{i \in \mathcal{I}} [M_{\mathsf{NL}}]_i \\ [\pi]_i \coloneqq \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, [M_{\mathsf{NL}}]_i),\ \forall i \in [1,m] \\ [\overline{\pi}]_{i^*} \coloneqq [\pi]_{i^*} \\ \overline{\pi} \coloneqq \sum_{i=1}^m [\overline{\pi}]_i \\ \epsilon \coloneqq \sum_{i=1}^m [\overline{\pi}]_i - [\pi]_i \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$, $M_{\mathsf{NL}} \leftarrow \mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, x, y, w)$, $\mathcal{I} \subset [1,m]$ *is a set of $m-1$ corrupt provers and $i^* \notin \mathcal{I}$ is the index of the honest prover.*

Proof. Robust verification in the distributed prover setting follows from robust verification of the Splittable zkSNARG since the adversary's view in the experiment is independent of M_{NL} and hence the proof π. More formally, let the output of the experiment be defined as 1 when $\mathsf{C}(x,w) = y$, $\epsilon \neq 0$ but Verify outputs 1, and be defined as 0 otherwise. We use a hybrid argument to show that for any polynomial size adversary $\mathcal{A}$, the output of the experiment is 1 with at most negligible probability.

- Hyb_0: This corresponds to the output of the experiment when run with adversary $\mathcal{A}$.
- Hyb_1: In this hybrid, the experiment computes $\overline{\pi} \coloneqq \pi + \epsilon$ where $\pi \coloneqq \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{NL}})$ and $\epsilon \coloneqq \sum_{i=1}^m [\overline{\pi}]_i - [\pi]_i$.

  Observe that in Hyb_0, $[M_{\mathsf{NL}}]$ is an additive sharing of $M_{\mathsf{NL}} \leftarrow \mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, x, y, w)$. The fact that $\mathsf{Prove}_{\mathsf{L}}$ is a group homomorphism implies that $[\pi]$ computed in Hyb_0 is an additive sharing of the proof $\pi \coloneqq \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{NL}})$ computed in this hybrid. We have

  $$\overline{\pi} = \sum_{i=1}^m [\overline{\pi}]_i = \pi + \sum_{i=1}^m [\overline{\pi}]_i - [\pi]_i = \pi + \epsilon$$

  where the first equality follows from the definition of $\overline{\pi}$, the second equality follows from the fact that $[\pi]$ constitutes an additive sharing of π and the third equality follows from the definition of ϵ. It follows that $\mathsf{Hyb}_0 \stackrel{\text{P}}{=} \mathsf{Hyb}_1$.
- Hyb_2: In this hybrid, the experiment computes $\overline{\pi} \coloneqq \pi' + \epsilon$ where $\pi' \leftarrow \mathsf{Prove}(\mathsf{crs}, x, y, w)$ and ϵ is computed as in the previous hybrid.

  $\mathsf{Hyb}_1 \stackrel{\text{c}}{\approx} \mathsf{Hyb}_2$ since the split prover property of SNARG implies that $\overline{\pi}$ computed in Hyb_1 is computationally indistinguishable from $\pi' + \epsilon$.
- Hyb_3: In this hybrid, the experiment always outputs 0.

  Observe that Hyb_2 is 1 only when $\epsilon \neq 0$ but $\overline{\pi}$ verifies successfully. We show that this occurs with at most negligible probability by a reduction to the robust verification property of SNARG. Consider an adversary $\mathcal{A}'$ against the robust verification of SNARG that does the following. It first runs $\mathcal{A}$ to

obtain $(x, y, w, \mathcal{I}, [M_{\mathsf{NL}}]_{\mathcal{I}}, [\overline{\pi}]_{\mathcal{I}})$ and then computes $[\pi]_{\mathcal{I}} \coloneqq \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, [M_{\mathsf{NL}}]_{\mathcal{I}})$. It then computes $\epsilon \coloneqq \sum_{i \in \mathcal{I}} [\overline{\pi}]_i - [\pi]_i$ and outputs (x, y, w, ϵ).

Since $[\overline{\pi}]_{i^*} = [\pi]_{i^*}$ for the honest prover i^*, ϵ computed in the experiment in Hyb_2 is identical to the one computed by $\mathcal{A}'$. Thus, if Hyb_2 is 1 then $\mathcal{A}'$ wins the robust verification experiment. It follows from the robust verification property of SNARG that Hyb_2 is 1 with at most negligible probability which in turn implies that $\mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3$.

It follows from our sequence of hybrids that $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_3$ and since Hyb_3 is always 0, any adversary $\mathcal{A}$ wins the DP-robust verification experiment with at most negligible probability. ■

4.2 The Groth16 zkSNARK

In this section we describe the Groth16 [Gro16] zkSNARK and show that it is a Splittable zkSNARK. We start by defining bilinear groups and Quadratic Arithmetic Programs (QAP), both of which will be important for describing the Groth16 zkSNARK.

Prime-order Bilinear Groups. Let $\mathcal{G}$ be a group generator such that $(p, \mathbb{G}_1, \mathbb{G}_T, e, g_1) \leftarrow \mathcal{G}(1^\lambda)$ be bilinear groups where $\mathbb{G}_1$ and $\mathbb{G}_T$ have prime order p and are equipped with efficient algorithms to carry out group operations, the pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ is an efficiently computable non-degenerate bilinear map, g_1 is a generator for $\mathbb{G}_1$ and $g_T = e(g_1, g_1)$ is a generator for $\mathbb{G}_T$. For any $\mathbf{x} = (x_1, \ldots, x_\ell) \in \mathbb{Z}_p^\ell$, we use $g_i^{\mathbf{x}} = (g_i^{x_1}, \ldots, g_i^{x_\ell})$ to denote the vector of group elements in $\mathbb{G}_i^\ell$ where $i \in \{1, T\}$.

Quadratic Arithmetic Programs. Given an arithmetic circuit consisting of addition and multiplication gates over a finite field $\mathbb{F}$, one can assign a set of input wires $\mathbf{x}$ and output wires $\mathbf{y}$ as specifying a statement $(\mathbf{x}, \mathbf{y})$ and all other wires in the circuit as defining a witness $\mathbf{w}$. This in turn defines a relation R consisting of values on statement and witness wires that satisfy the circuit. A *Quadratic Arithmetic Program (QAP)* [GGPR13] uses the *same* statement $(\mathbf{x}, \mathbf{y})$ and witness $\mathbf{w}$ but encodes the circuit satisfiability constraints as polynomials over $\mathbb{F}$. Specifically, given a circuit C, a QAP defines the relation

$$\mathcal{R}_{\mathsf{C}}^{\mathsf{qap}} = \left\{ (\mathbf{x}, \mathbf{y}; \mathbf{w}) \;\middle|\; \begin{array}{r} \mu_0 = 1 \\ (\mathbf{x}, \mathbf{y}) = (\mu_1, \ldots, \mu_{\ell_{\mathsf{stmt}}}) \in \mathbb{F}^{\ell_{\mathsf{stmt}}} \\ \mathbf{w} = (\mu_{\ell_{\mathsf{stmt}}+1}, \ldots, \mu_L) \in \mathbb{F}^{L-\ell_{\mathsf{stmt}}} \\ \sum_{i=0}^{L} \mu_i \rho_{1,i}(X) \cdot \sum_{i=0}^{L} \mu_i \rho_{2,i}(X) \equiv \sum_{i=0}^{L} \mu_i \rho_{3,i}(X) \bmod \rho_{\mathsf{mod}}(X) \end{array} \right\}$$

where $\rho_{\mathsf{mod}} \in \mathbb{F}[X]$ is of degree d and $\rho_{1,i}, \rho_{2,i}, \rho_{3,i} \in \mathbb{F}[X]$ are of degree strictly lesser than d for every $i \in [0, L]$. The polynomials $\rho_{1,i}, \rho_{2,i}, \rho_{3,i}, \rho_{\mathsf{mod}}$ encode the satisfiability constraints for C while $(\mathbf{x}, \mathbf{y})$ and $\mathbf{w}$ correspond to values on the statement and witness wires as described previously. We refer to $\mu_0 \| \mathbf{x} \| \mathbf{y} \| \mathbf{w} = (\mu_0, \ldots, \mu_L)$ as the extended witness of the QAP instance.

Remark 2 (QAPs For Boolean Circuits)**.** *We note that a QAP over any finite field $\mathbb{F}$, can be used to encode circuit satisfiability constraints for boolean circuits. Specifically, given a boolean circuit C, we can simulate C using an arithmetic circuit $\mathsf{C}_{\mathbb{F}}$ over $\mathbb{F}$ by mapping the 0 bit input to the additive identity $0_{\mathbb{F}}$, the 1 bit input to the multiplicative identity $1_{\mathbb{F}}$, and computing each NAND gate with inputs a and b in C as $1 - ab$ in $\mathsf{C}_{\mathbb{F}}$. We can then encode satisfiability constraints for $\mathsf{C}_{\mathbb{F}}$ using a QAP over $\mathbb{F}$.*

We recall the zkSNARK presented by Groth [Gro16] in Figure 2, which we denote by Groth16. While Groth16 is a zkSNARK for QAP relations, we present it as a zkSNARK for boolean circuit satisfiability for ease of use in our main constructions. Our primary goal is to demonstrate that it is a Splittable zkSNARK and so we omit a number of details in order to provide a simpler description of the scheme that suffices for our purposes. We refer the reader to [Gro16] for a complete description. While [Gro16] uses asymmetric

The Groth16 zkSNARK

Public parameters. A group generator $\mathcal{G}$ for prime-order bilinear groups.

Groth16.Setup$(1^\lambda, \mathsf{C})$

1: Takes a boolean circuit $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ as input, runs $(p, \mathbb{G}_1, \mathbb{G}_T, e, g_1) \leftarrow \mathcal{G}(1^\lambda)$ and outputs a common reference string crs consisting of the description of the bilinear group and a vector of elements in $\mathbb{G}_1$ of the form

$$\begin{pmatrix} Q_{a,0}, \ldots, Q_{a,L+1} \\ Q_{b,0}, \ldots, Q_{b,L+1} \\ Q_{c,0}, \ldots, Q_{c,L+d-1} \\ Q_{v,1}, Q_{v,2} \end{pmatrix}.$$

Groth16.Prove$(\mathsf{crs}, \mathbf{x}, \mathbf{y}, \mathbf{w})$

1: Parse $(\mathbf{x}, \mathbf{y}) = (\mu_1, \ldots, \mu_{\ell_{\mathsf{stmt}}})$.

2: Compute the extended witness $(\mu_1, \ldots, \mu_L)$ for the corresponding QAP instance.

3: Compute the polynomial $\rho_{\mathsf{div}}(X)$ of degree at most $d-2$ such that

$$\rho_{\mathsf{div}}(X) = \frac{\sum_{j,k \in [0,L]} \rho_{1,j}(X) \cdot \mu_j \mu_k \cdot \rho_{2,k}(X) - \sum_{j=0}^{L} \mu_j \rho_{3,j}(X)}{\rho_{\mathsf{mod}}(X)}. \tag{1}$$

where $\rho_{1,i}(X), \rho_{2,i}(X), \rho_{3,i}(X)$ for every $i \in [0, L,]$ and $\rho_{\mathsf{mod}}(X)$ are in $\mathbb{Z}_p[X]$, defined by $\mathcal{R}_{\mathsf{C}}^{\mathsf{qap}}$.

4: Sample $r, s \leftarrow \mathbb{Z}_p$ uniformly at random and compute the proof $\boldsymbol{\pi} = (\pi_a, \pi_b, \pi_c)$ as

$$\pi_a = Q_{a,L+1} \cdot Q_{v,2}^r \cdot \prod_{i=0}^{L} Q_{a,i}^{\mu_i}$$

$$\pi_b = Q_{b,L+1} \cdot Q_{v,2}^s \cdot \prod_{i=0}^{L} Q_{b,i}^{\mu_i}$$

$$\pi_c = \frac{\pi_a^s \cdot \pi_b^r \cdot \prod_{i=\ell_{\mathsf{stmt}}+1}^{L} Q_{c,i}^{\mu_i} \cdot \prod_{i=0}^{d-2} Q_{c,L+1+i}^{\rho_{\mathsf{div}}[i]}}{Q_{v,2}^{rs}}$$

where $\rho_{\mathsf{div}}[i]$ denotes the i-th co-efficient of $\rho_{\mathsf{div}}(X)$.

Groth16.Verify$(\mathsf{crs}, \mathbf{x}, \mathbf{y}, \boldsymbol{\pi})$

1: Parse $(\mathbf{x}, \mathbf{y}) = (\mu_1, \ldots, \mu_{\ell_{\mathsf{stmt}}})$ and $\boldsymbol{\pi} = (\pi_a, \pi_b, \pi_c)$.

2: Output 1 if

$$e(\pi_a, \pi_b) = e\big(Q_{a,L+1}, Q_{b,L+1}\big) \cdot e\left(\prod_{i=0}^{\ell_{\mathsf{stmt}}} Q_{c,i}^{\mu_i}, Q_{v,1}\right) \cdot e\big(\pi_c, Q_{v,2}\big)$$

and outputs 0 otherwise.

Figure 2: The Groth16 zkSNARK [Gro16].

bilinear groups for the construction, our presentation uses symmetric bilinear groups for ease of exposition. We note that the scheme is secure even in this case [Gro16]. Additionally, our arguments in favor of Groth16 being Splittable when instantiated with symmetric bilinear groups extends directly to the case when it is instantiated with asymmetric bilinear groups too. The following theorem immediately follows from the fact that Groth16 is a zkSNARK and we show in Theorem 9 that it is additionally a Splittable zkSNARK.

Theorem 8 ([Gro16]). *Groth16 (Figure 2) is a zkSNARK for boolean circuit satisfiability in the generic bilinear group model.*

Theorem 9. *Groth16 (Figure 2) is a Splittable zkSNARK for boolean circuit satisfiability (Definition 5) in the generic bilinear group model. Moreover, for every polynomial size boolean circuit* C *of depth* D*, there exists an efficiently constructible polynomial size circuit* C_{NL} *over* $\mathbb{Z}_p$ *with multiplicative depth* $O(D)$ *such that for all* crs *in the support of* Setup*, all* $(\mathsf{C}, \mathbf{x}, \mathbf{y}; \mathbf{w}) \in \mathsf{SAT}$ *and all* $\mathbf{r} \in \mathbb{Z}_p^2$*, we have* $\mathsf{C}_{\mathsf{NL}}(\mathbf{x}, \mathbf{w}, \mathbf{r}) = \mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, \mathbf{x}, \mathbf{y}, \mathbf{w}; \mathbf{r})$*, where* p *is the order of the bilinear group.*

Proof. We first show that Groth16 is split prover and then argue that it has robust verification.

Claim 9.1. *Groth16 is split prover such that for every polynomial size boolean circuit* C *there exists an efficiently constructible circuit* C_{NL} *as described above.*

Proof. Consider the following split of the Groth16 prover into a non-linear and linear phase.

- $\mathsf{Groth16.Prove}_{\mathsf{NL}}$: It takes takes the statement $(\mathbf{x}, \mathbf{y})$, witness $\mathbf{w}$ and uniform random values $r, s \leftarrow \mathbb{Z}_p$ as input and outputs the following.
  1. The extended witness $(1, \mu_0, \ldots, \mu_L)$ for the corresponding QAP relation.
  2. The random inputs, their product, and the product of each random input with the extended witness $(r, s, rs, r\mu_1, \ldots, r\mu_L, s\mu_1, \ldots, s\mu_L)$.
  3. The co-efficients of the polynomial $\rho_{\mathsf{div}}(X)$, defined in Equation (1).
- $\mathsf{Groth16.Prove}_{\mathsf{L}}$: It takes the output of the non-linear phase as input and computes $\boldsymbol{\pi} = (\pi_a, \pi_b, \pi_c)$ as

$$\pi_a = Q_{a,L+1} \cdot Q_{v,2}^{r} \cdot \prod_{i=0}^{L} Q_{a,i}^{\mu_i}$$

$$\pi_b = Q_{b,L+1} \cdot Q_{v,2}^{s} \cdot \prod_{i=0}^{L} Q_{b,i}^{\mu_i}$$

$$\pi_c = Q_{v,2}^{rs} \cdot \left(Q_{a,L+1}^{s} \cdot \prod_{i=0}^{L} Q_{a,i}^{s\mu_i}\right) \cdot \left(Q_{b,L+1}^{r} \cdot \prod_{i=0}^{L} Q_{b,i}^{r\mu_i}\right) \cdot \prod_{i=\ell_{\mathsf{stmt}}+1}^{L} Q_{c,i}^{\mu_i} \cdot \prod_{i=0}^{d-2} Q_{c,L+1+i}^{\rho_{\mathsf{div}}[i]}$$

  where we highlight the output of the non-linear phase in red for clarity.

It is easy to see that $\mathsf{Groth16.Prove}_{\mathsf{L}}$ is a homomorphism from tuples in $\mathbb{Z}_p$ to tuples in $\mathbb{G}_1$. The split prover property then follows from the fact that $\mathsf{Groth16.Prove}_{\mathsf{NL}}$ and $\mathsf{Groth16.Prove}_{\mathsf{L}}$ constitute a partition of the steps of Groth16.Prove.

Given any polynomial size boolean circuit C, we next describe the circuit C_{NL} that computes the output of the non-linear phase when proving satisfiability of C. As discussed in Remark 2, the extended witness for C corresponds to the value on each wire of the circuit $\mathsf{C}_{\mathbb{Z}_p}$ that emulates C over $\mathbb{Z}_p$. Thus, the extended witness can be computed by a circuit over $\mathbb{Z}_p$ with multiplicative depth $O(D)$. Note that the *boolean*

statement and witness are implicitly mapped to $0, 1 \in \mathbb{Z}_p$ when used as input to C_{NL}, as described in Section 3. Next, observe that given the extended witness, each value in $(rs, r\mu_1, \ldots, r\mu_L, s\mu_1, \ldots, s\mu_L)$ can be computed in parallel and only increments the multiplicative depth by 1. Finally, to compute the co-efficients of $\rho_{\mathsf{div}}(X)$, it suffices to first evaluate the right-hand side of Equation (1) at $d-1$ distinct points that do not contain a root of $\rho_{\mathsf{mod}}(X)$, and then reconstruct the co-efficients from these evaluations. Since the polynomials $\{\rho_{1,i}, \rho_{2,i}, \rho_{3,i}\}_{i=1}^{L}$ are completely defined by C and are independent of the statement and witness, their evaluations at $d-1$ distinct points can be hardcoded into C_{NL}. Thus, each co-efficient of $\rho_{\mathsf{div}}(X)$ can be computed as a linear combination of the pairwise product of the extended witness $\{\mu_i\mu_j\}_{i,j}$. Since computing $\{\mu_i\mu_j\}_{i,j}$ requires evaluating an additional layer of multiplication gates, it follows that C_{NL} is a polynomial size circuit over $\mathbb{Z}_p$ with a multiplicative depth of $O(D)$. Moreover, it follows from our description of C_{NL} and the fact that the degree of $\rho_{\mathsf{mod}}(x)$ and each $\rho_{1,i}, \rho_{2,i}, \rho_{3,i}$ is polynomial in $|\mathsf{C}|$ that C_{NL} is efficiently constructible given C. □

Claim 9.2. *Groth16 has robust verification.*

Proof. Consider an adversary $\mathcal{A}$ against the robust verification property of Groth16 that outputs a statement $(\mathbf{x}, \mathbf{y})$, witness $\mathbf{w}$ and an error $\boldsymbol{\epsilon} = (g_1^{\epsilon_a}, g_1^{\epsilon_b}, g_1^{\epsilon_c})$ where ϵ_a, ϵ_b and ϵ_c are in $\mathbb{Z}_p$. Let the proof $\boldsymbol{\pi}$ be of the form $\boldsymbol{\pi} = (g_1^{z_a}, g_1^{z_b}, g_1^{z_c})$. Since we are using a multiplicative group, we have $\overline{\boldsymbol{\pi}} = (g_1^{z_a+\epsilon_a}, g_1^{z_b+\epsilon_b}, g_1^{z_c+\epsilon_c})$. We argue that when $\mathsf{C}(\mathbf{x}, \mathbf{w}) = \mathbf{y}$ and at least one among ϵ_a, ϵ_b and ϵ_c is non-zero then $\mathsf{Groth16.Verify}(\mathsf{crs}, \mathbf{x}, \mathbf{y}, \mathbf{w}, \overline{\boldsymbol{\pi}}) = 1$ with at most negligible probability. Informally, our argument relies on the fact that $r, s \leftarrow \mathbb{Z}_p$ sampled uniformly at random for zero-knowledge of Groth16 also ensure robust verification.

From the completeness of Groth16 we have

$$e\left(g_1^{z_a}, g_1^{z_b}\right) = e\left(Q_{a,L+1}, Q_{a,L+1}\right) \cdot e\left(\prod_{i=0}^{\ell_{\mathsf{stmt}}} Q_{c,i}^{\mu_i}, Q_{v,1}\right) \cdot e\left(g_1^{z_c}, Q_{v,2}\right) \tag{2}$$

with overwhelming probability where $(\mathbf{x}, \mathbf{y}) = (\mu_1, \ldots, \mu_{\ell_{\mathsf{stmt}}})$. Now, observe that

$$\begin{aligned}
&\mathsf{Groth16.Verify}(\mathsf{crs}, \mathbf{x}, \mathbf{y}, \overline{\boldsymbol{\pi}}) = 1 \\
\implies &e\left(g_1^{z_a+\epsilon_a}, g_1^{z_b+\epsilon_b}\right) = e\left(Q_{a,L+1}, Q_{b,L+1}\right) \cdot e\left(\prod_{i=0}^{\ell_{\mathsf{stmt}}} Q_{c,i}^{\mu_i}, Q_{v,1}\right) \cdot e\left(g_1^{z_c+\epsilon_c}, Q_{v,2}\right) \\
\implies &g_T^{z_a\epsilon_b} \cdot g_T^{z_b\epsilon_a} \cdot g_T^{\epsilon_a\epsilon_b} = e\left(g_1^{\epsilon_c}, Q_{v,2}\right) \\
\implies &g_T^{z_a\epsilon_b + z_b\epsilon_a + \epsilon_a\epsilon_b - \epsilon_c\, q_{v,2}} = 1
\end{aligned}$$

holds with overwhelming probability where $Q_{v,2} = g_1^{q_{v,2}}$ and the second implication follows from Equation (2) and the definition of $\overline{\boldsymbol{\pi}}$.

Since r and s are uniformly random over $\mathbb{Z}_p$ and independent of $\mathcal{A}$'s view, z_a and z_b are uniformly random over $\mathbb{Z}_p$ and are independent of $\boldsymbol{\epsilon}$. Therefore, when either $\epsilon_a \neq 0$ or $\epsilon_b \neq 0$ then from the Schwartz-Zippel lemma, $z_a\epsilon_b + z_b\epsilon_a - \epsilon_c\, q_{v,2} + \epsilon_a\epsilon_b = 0$ over $\mathbb{Z}_p$ with probability at most $1/|\mathbb{Z}_p| = 1/p \in \mathsf{negl}(\lambda)$. Moreover, $q_{v,2} \neq 0$ by definition [Gro16] which implies that $\epsilon_c\, q_{v,2} = 0$ only when $\epsilon_c = 0$. Thus, $g_T^{z_a\epsilon_b + z_b\epsilon_a + \epsilon_a\epsilon_b - \epsilon_c\, q_{v,2}} = 1$ with at most negligible probability when at least one among ϵ_a, ϵ_b or ϵ_c is non-zero which implies that $\mathsf{Groth16.Verify}(\mathsf{crs}, \mathbf{x}, \mathbf{y}, \overline{\boldsymbol{\pi}}) = 1$ with at most negligible probability in this case. □

Since $\mathsf{Groth16.Prove}_{\mathsf{NL}}$ only requires two uniformly random value from $\mathbb{Z}_p$, each of which is $O(\log \lambda)$ bits long, it follows from Theorem 8 and Claims 9.1 and 9.2 that Groth16 is a Splittable zkSNARG for boolean circuit satisfiability in the generic bilinear group model. ■

Theorem 9 implies that, in terms of circuit depth, the non-linear phase for Groth16 is as easy to compute as the circuit C defined by the statement. Since many HSS schemes only support evaluating circuits with bounded depth, this ensures that for any function computed within HSS, the corresponding non-linear phase for proving correctness of the computation can also be evaluated within the same HSS scheme[16]. Moreover, observe that the non-linear phase, as described in the proof of Theorem 9, does not require performing any group operations. Consequently, describing C_{NL} does not require making non-black-box use of the group.

Remark 3 (Beyond boolean satisfiability)**.** *While we will primarily focus on boolean circuit satisfiability, the proof of Theorem 9 extends naturally to a stronger result: for any circuit* C *over* $\mathbb{Z}_p$ *with multiplicative depth* D*, the non-linear phase for proving satisfiability of* C *can be computed by a circuit* C_{NL} *over* $\mathbb{Z}_p$ *with multiplicative depth* $O(D)$*. In fact, if* C *is a polynomial size RMS program with modulus* p*, then it can be shown that* C_{NL} *is also a polynomial size RMS program with modulus* p*. Since a depth* D *computation over* $\mathbb{Z}_p$ *might incur a multiplicative overhead of* $O(\log \lambda)$ *when expressed as a boolean circuit, this allows evaluating a broader class of functions without increasing the complexity of the non-linear phase.*

4.3 The Waters-Wu zkBARG

In this section we recall the Waters-Wu Batch Argument (BARG) [WW22] and discuss a variant of the construction that achieves perfect zero-knowledge. We then proceed to show that the zero-knowledge variant of the Waters-Wu BARG is Splittable. We start by defining composite-order bilinear groups and then continue to discuss the details of the Waters-Wu BARG.

Composite-order Bilinear Groups. Let $\mathcal{G}$ be a group generator such that $(p, q, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda)$ consists of two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ of order $N = pq$ where p, q are primes of magnitude $\Theta(2^\lambda)$, a generator g for $\mathbb{G}$ and an efficiently computable pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ that is a non-degenerate bilinear map. Let $g_T = e(g, g)$ be the generator of $\mathbb{G}_T$. By the Chinese Remainder Theorem, we can write $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q$ where $\mathbb{G}_p$ is a subgroup of order p (generated by $g_p = g^q$) and $\mathbb{G}_q$ is a subgroup of order q (generated by $g_q = g^p$). The subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$ are orthogonal i.e., $e(g_p, g_q) = 1$, where 1 denotes the identity element in $\mathbb{G}_T$.

Definition 6 (Subgroup Decision Assumption [BGN05])**.** *The subgroup decision assumption holds with respect to a composite-order bilinear group generator* $\mathcal{G}$ *if for every polynomial size adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr\left[\mathcal{A}\left((\mathbb{G}, \mathbb{G}_T, N, g_p, e), g^r\right) = 1\right] - \Pr\left[\mathcal{A}\left((\mathbb{G}, \mathbb{G}_T, N, g_p, e), g_p^r\right) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where $(p, q, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda)$*,* $N = pq$*,* $g_p = g^q$*, and* $r \leftarrow \mathbb{Z}_N$*.*

We present the zero-knowledge variant of the Waters-Wu BARG in Figure 3. Ensuring zero-knowledge only requires modifying the prover's algorithm and retains the same setup and verification algorithms as in [WW22]. We first show that the construction in Figure 3 satisfies perfect zero-knowledge while retaining the other properties of the Waters-Wu BARG. We then continue to show that it is a Splittable zkSNARG.

Theorem 10. *Assuming the subgroup decision assumption holds with respect to composite-order bilinear groups, the Waters-Wu zkBARG (Figure 3) is a batch argument for boolean circuit satisfiability with non-adaptive soundness and perfect zero-knowledge.*

[16] This requires that the HSS scheme supports computing over $\mathbb{Z}_p$, which is true for the schemes we consider.

Waters-Wu zkBARG

Public parameters A group generator $\mathcal{G}$ for composite-order bilinear groups.

zkBARG.Setup$(1^\lambda, n)$

1: Run $(p, q, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda)$ and let $N = pq$, and $g_p = g^q$.
2: Sample $\alpha_i \leftarrow \mathbb{Z}_N$ uniformly at random for each $i \in [1, n]$.
3: Let $A_i = g_p^{\alpha_i}$ for each $i \in [1, n]$ and let $A = \prod_{i\in[1,n]} A_i = g_p^{\sum_{i\in[1,n]} \alpha_i}$.
4: Let $B_{i,j} = g_p^{\alpha_i \alpha_j}$ for every $i, j \in [1, n]$ such that $i \neq j$.
5: Output $\mathsf{crs} = \left((\mathbb{G}, \mathbb{G}_T, N, g_p, e), A, \{A_i\}_{i=1}^m, \{B_{i,j}\}_{i\neq j\in[1,m]}\right)$.

zkBARG.Prove$(\mathsf{crs}, \mathsf{C}, (\mathbf{x}_1, \ldots, \mathbf{x}_n), (\mathbf{w}_1, \ldots, \mathbf{w}_n))$

Let L be the number of wires in C and let $\boldsymbol{\mu}_i = (\mu_{i,1}, \ldots, \mu_{i,L})$ be the i-th extended witness where $\mu_{i,j}$ is the value on the j-th wire in $\mathsf{C}(\mathbf{x}_i, \mathbf{w}_i)$ for every $i \in [1, n]$ and $j \in [1, L]$. For every $j \in [1, \ell_{\mathsf{stmt}}]$ we have $\mu_{i,j} = x_j$.

1: **Commitment To Wire Values:** For each $k \in [1, L]$, sample $r_k \leftarrow \mathbb{Z}_N$ uniformly at random if $k > \ell_{\mathsf{stmt}}$ and set $r_k := 0$ if $k \leq \ell_{\mathsf{stmt}}$ and compute

$$U_k = g_p^{r_k} \cdot \prod_{i\in[1,n]} A_i^{\mu_{i,k}} = g_p^{r_k + \sum_{i\in[1,n]} \alpha_i \mu_{i,k}}.$$

2: **Validity Of Wire Assignments:** For each $k \in [1, L]$ compute

$$V_k = \frac{\prod_{i\neq j} B_{i,j}^{(1-\mu_{i,k})\mu_{j,k}} \cdot \prod_{i\in[1,n]} A_i^{r_k(1-2\mu_{i,k})}}{g_p^{r_k^2}}.$$

3: **Validity Of Gate Computations:** For every $k \in [1, s]$, where the k-th NAND gate has input wires (a, b) and output wire c, compute

$$W_k = \frac{\prod_{i\neq j} B_{i,j}^{\mu_{i,a}\mu_{j,b} + \mu_{j,c} - 1} \cdot A^{r_c} \cdot U_b^{r_a} \cdot U_a^{r_b}}{g_p^{r_a r_b}}$$

4: Output the proof $\boldsymbol{\pi} = \left(\{U_k\}_{i=1}^L, \{V_k\}_{i=1}^L, \{W_k\}_{i=1}^s\right)$.

Figure 3: Setup and prover algorithms for the zero-knowledge variant of the Waters-Wu BARG [WW22]. The verification algorithm is described in Figure 4. Changes to the prover algorithm from [WW22], required to achieve zero-knowledge, are highlighted in red.

Waters-Wu zkBARG (continued)

zkBARG.Verify(crs, C, $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\boldsymbol{\pi}$)

1: **Verification Key:** For each $k \in [1, \ell_{\mathsf{stmt}}]$ compute

$$U_k^* = \prod_{i \in [1,n]} A_i^{x_{i,k}} = g_p^{\sum_{i \in [1,n]} \alpha_i x_{i,k}}.$$

2: **Statement Consistency Check:** For each $k \in [1, \ell_{\mathsf{stmt}}]$, check if

$$U_k = U_k^*.$$

3: **Wire Validity Checks:** For each $k \in [\ell_{\mathsf{stmt}} + 1, L]$, check if

$$e(A, U_k) = e(g_p, V_k) \cdot e(U_k, U_k).$$

4: **Gate Consistency Checks:** For each $k \in [1, s]$, where the k-th NAND gate has input wires (a, b) and output wire c, check if

$$e(A, A) \cdot e(g_p, W_k) = e(U_a, U_b) \cdot e(A, U_c).$$

5: Output 1 if all checks pass and output 0 otherwise.

Figure 4: Verification algorithm for the zero-knowledge variant of the Waters-Wu BARG [WW22]. The setup and prover algorithms are described in Figure 3.

Proof. It is easy to see from inspection that modifications to achieve zero-knowledge do not affect completeness. Moreover, observe that the Setup and Verify algorithms in Figure 3 remain identical to those in the Waters-Wu BARG [WW22]. It follows that Figure 3, like the Waters-Wu BARG, is non-adaptively sound.

We are left to argue that the scheme has perfect zero-knowledge. We first describe the simulator and then argue that a simulated proof is identically distributed to a proof computed using the zkBARG. The simulator works as follows.

1. It runs the setup algorithm and outputs crs and st $= \left(g_p, N, \{\alpha_i\}_{i=1}^n\right)$.
2. When invoked with input $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and st, it does the following.
   - *Commitments To Wire Values:* For every $k \in [1, \ell_{\mathsf{stmt}}]$, it sets $z_k = \sum_{i \in [1,m]} \alpha_i x_{i,k}$. For the remaining wires $k \in [\ell_{\mathsf{stmt}} + 1, L]$, it samples $z_k \leftarrow \mathbb{Z}_N$ uniformly at random. For all $k \in [1, L]$, it computes $U_k := g_p^{z_k}$.
   - *Validity Of Wire Assignments:* For each $k \in [1, L]$, it sets $V_k = g_p^{y_k}$ where

$$y_k = \left(z_k \sum_{i \in [1,n]} \alpha_i\right) - z_k^2$$

   - *Gate Consistency Check:* For each $k \in [1, s]$, where the k-th NAND gate has input wires (a, b) and

output wire c, it sets $W_k = g_p^{p_k}$, where

$$p_k = z_a z_b + \left(z_c \sum_{i \in [1,n]} \alpha_i \right) - \sum_{i \in [1,n]} \alpha_i^2 - \sum_{i \neq j} \alpha_i \alpha_j.$$

– It outputs $\boldsymbol{\pi} = \left(\{U_k\}_{k=1}^{L}, \{V_k\}_{k=1}^{L}, \{W_k\}_{k=1}^{s} \right)$.

Next, we argue that the joint distribution of the CRS and the proof output by the simulator is identical to that computed using the zkBARG. Observe that the distribution of the CRS is the same in both cases since the simulator simply runs Setup. Moreover, the proofs are identically distributed conditioned on the CRS. Specifically, for each input wire $k \in [1, \ell_{\mathsf{stmt}}]$, U_k is identical in the real and simulated proofs. For each $k \in [\ell_{\mathsf{stmt}} + 1, L]$, U_k computed in Prove is uniformly random in $\mathbb{G}_p$ since r_k is sampled uniformly at random from $\mathbb{Z}_N$. Similarly, U_k output by the simulator is also uniformly random in $\mathbb{G}_p$ and thus it follows that $\{U_k\}_{k=1}^{L}$ is identically distributed in the real and simulated proofs. Now, observe that each V_k and W_k are uniquely determined in $\mathbb{G}_p$ by the verification checks $e(A, U_k) = e(g_p, V_k) \cdot e(U_k, U_k)$ and $e(A, A) \cdot e(g_p, W_k) = e(U_a, U_b) \cdot e(A, U_c)$ respectively. Since the proof output by both Prove and the simulator are in $\mathbb{G}_p$, and the simulated proof passes the verification checks, it follows that the simulated proof is identically distributed to the real proof. Thus, the zkBARG has perfect zero-knowledge. ■

Remark 4 (Somewhere Argument Of Knowledge)**.** *As observed in the proof of Theorem 10, ensuring zero-knowledge only requires changing the prover's algorithm. This means that the zkBARG achieves any notion of soundness achieved by the Waters-Wu BARG. Specifically, this means that the zkBARG is a somewhere argument of knowledge [CJJ22].*

Theorem 11. *Assuming the subgroup decision assumption holds with respect to composite-order bilinear groups, the Waters-Wu zkBARG (Figure 3) is a Splittable zkSNARG (Definition 5), with non-adaptive soundness, for SIMD boolean circuit satisfiability. Moreover, for every polynomial size boolean circuit* C *of depth* D *and all positive integers* n*, there exists an efficiently constructible circuit* C_{NL} *over* $\mathbb{Z}_N$ *of size* $|\mathsf{C}_{\mathsf{NL}}| = O(|C|n)$ *and with multiplicative depth* $O(D)$ *such that for all* crs *in the support of* Setup*,* $(\mathsf{C}, \mathbf{x}_1, \ldots, \mathbf{x}_n; \mathbf{w}_1, \ldots, \mathbf{w}_n) \in \mathsf{BSAT}_n$ *and* $\mathbf{r} \in \mathbb{Z}_N^L$*, we have* $\mathsf{C}_{\mathsf{NL}}(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{w}_1, \ldots, \mathbf{w}_n, \mathbf{r}) = \mathsf{Prove}_{\mathsf{NL}}(\mathsf{crs}, \mathsf{C}, \mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{w}_1, \ldots, \mathbf{w}_n; \mathbf{r})$*, where* L *is the number of wires in* C*, and* N *is the order of the bilinear groups specified by* crs*.*

Proof. We first show that the zkBARG is split prover and then argue that it has robust verification.

Claim 11.1. *Waters-Wu zkBARG is split prover such that for every polynomial size boolean circuit* C *there exists an efficiently constructible circuit* C_{NL} *as described above.*

Proof. Consider the following split of the prover into a non-linear and linear phase.

– $\mathsf{zkBARG.Prove}_{\mathsf{NL}}$: It takes takes the statements $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, witnesses $(\mathbf{w}_1, \ldots, \mathbf{w}_n)$ and uniform random values $r_1, \ldots, r_L \leftarrow \mathbb{Z}_N$ as input and outputs the following.

1. The extended witness $\{\mu_{i,j}\}_{i,j}$, where $\mu_{i,j}$ is the value on the j-th wire of $\mathsf{C}(\mathbf{x}_i, \mathbf{w}_i)$, for each $i \in [1, n]$ and $j \in [1, L]$.
2. The pairwise product of the values in the extended witness, $\mu_{i,j,k_1,k_2} \coloneqq \mu_{i,k_1} \cdot \mu_{j,k_2}$ for every $i, j \in [1, n]$ such that $i \neq j$ and every $k_1, k_2 \in [1, L]$.
3. The random inputs $\{r_i\}_i$ and the pairwise product of the random inputs, $r_{i,j} \coloneqq r_i \cdot r_j$ for every $i, j \in [1, L]$.

4. The product of each random input with the extended witness, $\rho_{i,k_1,k_2} := \mu_{i,k_1} \cdot r_{k_2}$ for every $i \in [1,n]$ and $k_1, k_2 \in [1,L]$.

– zkBARG.Prove$_{\mathsf{L}}$: It takes the output of the non-linear phase (highlighted in red below) as input and proceeds as follows.

1. For each $k \in [1,L]$, it computes the commitment to wire values as

$$U_k = g_p^{r_k} \cdot \prod_{i \in [1,n]} A_i^{\mu_{i,k}}.$$

2. For each $k \in [1,L]$, it computes

$$V_k = \prod_{i \neq j} B_{i,j}^{\mu_{j,k}} \cdot \prod_{i \neq j} B_{i,j}^{\mu_{i,j,k,k}} \cdot \prod_{i \in [1,n]} A_i^{r_k} \cdot \prod_{i \in [1,n]} A_i^{2\rho_{i,k,k}} \cdot g_p^{-r_{k,k}}.$$

3. For each $k \in [1,s]$, where the k-th NAND gate has input wires (a,b) and output wire c, it computes

$$W_k = \prod_{i \neq j} B_{i,j}^{\mu_{i,j,a,b}} \cdot \prod_{i \neq j} B_{i,j}^{\mu_{j,c}} \cdot B_{i,j}^{-1} \cdot g_p^{-r_{a,b}} \cdot A^{r_c} \cdot \prod_{i \in [1,n]} A_i^{\rho_{i,a,b}} \cdot \prod_{i \in [1,n]} A_i^{\rho_{i,b,a}}.$$

It is easy to see that zkBARG.Prove$_{\mathsf{L}}$ is a homomorphism from tuples in $\mathbb{Z}_N$ to tuples in $\mathbb{G}$. The split prover property then follows from the fact that zkBARG.Prove$_{\mathsf{NL}}$ and zkBARG.Prove$_{\mathsf{L}}$ constitute a partition of the steps of zkBARG.Prove.

Given any polynomial size circuit C of depth D, we next show that there exists a circuit C_{NL} over $\mathbb{Z}_N$ of multiplicative depth $O(D)$ that computes the output of the non-linear phase when proving batch satisfiability of C. Observe that $(\mu_{i,1}, \ldots, \mu_{i,L})$ can be computed in parallel for each $i \in [1,n]$ by a circuit over $\mathbb{Z}_N$ with multiplicative depth $O(D)$, since any two-input boolean gate can be computed using a constant number of $\mathbb{Z}_N$ operations. Given $\{\mu_{i,j}\}_{i,j}$, each μ_{i,j,k_1,k_2} can be computed in parallel by a constant depth circuit over $\mathbb{Z}_N$. Similarly, each r_i and ρ_{i,k_1,k_2} can be computed in parallel by a constant depth circuit over $\mathbb{Z}_N$ given $\{\mu_{i,j}\}_{i,j}$. Putting everything together, it follows that C_{NL} is a circuit over $\mathbb{Z}_N$ with multiplicative depth $O(D)$. It is also easy to see that C_{NL} is efficiently constructible given C. □

Claim 11.2. *The Waters-Wu zkBARG has robust verification assuming the subgroup decision assumption holds with respect to the composite-order bilinear groups.*

Proof. Let $\mathcal{A}$ be an adversary against the robust verification of the Waters-Wu zkBARG that outputs a statement $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, witness $(\mathbf{w}_1, \ldots, \mathbf{w}_n)$ and error

$$\boldsymbol{\epsilon} = \left(\{g^{\epsilon_{u_k}}\}_{k=1}^{L}, \{g^{\epsilon_{v_k}}\}_{k=1}^{L}, \{g^{\epsilon_{w_k}}\}_{k=1}^{s}\right)$$

where all ϵ_{u_k}, ϵ_{v_k} and ϵ_{w_k} are in $\mathbb{Z}_N$. Let the proof $\boldsymbol{\pi} = \left(\{U_k\}_{i=1}^{L}, \{V_k\}_{i=1}^{L}, \{W_k\}_{i=1}^{s}\right)$ which means that

$$\overline{\boldsymbol{\pi}} = \left(\{U_k \cdot g^{\epsilon_{u_k}}\}_{i=1}^{L}, \{V_k \cdot g^{\epsilon_{v_k}}\}_{i=1}^{L}, \{W_k \cdot g^{\epsilon_{w_k}}\}_{i=1}^{s}\right)$$

since we are using multiplicative notation for the group.

Recall from Section 4.3 that $\mathbb{G}$ is isomorphic to $\mathbb{G}_p \times \mathbb{G}_q$ and that the subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$ are orthogonal with respect to the pairing e which implies that given the generator g_p of $\mathbb{G}_p$, we can test if any element g' is in $\mathbb{G}_q$ by checking that $e(g_p, g') = 1_{\mathbb{G}_T}$.

Informally, the verification checks for the zkBARG pass with overwhelming probability only when errors to each term of the proof are in the subgroup $\mathbb{G}_q$. However, the robust verification property requires that $\overline{\boldsymbol{\pi}}$ not verify successfully when the error to any term of the proof is not $1_{\mathbb{G}}$. Thus, our proof proceeds in two steps. First, we show that it is computationally infeasible for the adversary to output an error $\boldsymbol{\epsilon}$ where one of the terms is in $\mathbb{G}_q$ but not equal to $1_{\mathbb{G}}$. This means that any error term output by $\mathcal{A}$ that is not equal to $1_{\mathbb{G}}$ cannot be in $\mathbb{G}_q$. Next, we show that verification of $\overline{\boldsymbol{\pi}}$ fails with overwhelming probability if there exists an error term that is not in $\mathbb{G}_q$ and not equal to $1_{\mathbb{G}}$. It then follows that the zkBARG has robust verification.

Let the output of the robust verification experiment be defined as 1 when $(\mathbf{w}_1, \ldots, \mathbf{w}_n)$ is a valid witness for $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, there exists a value in $\boldsymbol{\epsilon}$ that is not equal to $1_{\mathbb{G}}$ and zkBARG.Verify$(\mathsf{crs}, \mathbf{x}_1, \ldots, \mathbf{x}_n, \overline{\boldsymbol{\pi}}) = 1$; and be defined as 0 otherwise. We use a hybrid argument to show that the output of the experiment is 1 with at most negligible probability.

- Hyb_0: This corresponds to the output of the experiment when run with $\mathcal{A}$.
- Hyb_1: In this hybrid, the experiment outputs 0 if there exists $g^{\eta} \in \boldsymbol{\epsilon}$ such that $e(g_p, g^{\eta}) = 1_{\mathbb{G}_T}$.

  Observe that if every element in $\boldsymbol{\epsilon}$ is $1_{\mathbb{G}}$ then both Hyb_0 and Hyb_1 are equal to 0. However, Hyb_0 might not be the same as Hyb_1 if there exists $g^{\eta} \in \mathbb{G}_q$ such that $g^{\eta} \neq 1_{\mathbb{G}}$. We show that Hyb_1 is indistinguishable from Hyb_0 by reduction to the subgroup decision assumption. Specifically, consider an adversary $\mathcal{A}'$ for the subgroup decision problem that upon receiving $\left(\left(\mathbb{G}, \mathbb{G}_T, N, g_p, e\right), g'\right)$ computes the crs for the zkBARG and runs $\mathcal{A}(1^{\lambda}, \mathsf{crs})$ to obtain $\boldsymbol{\epsilon}$. If there exists g^{η} in $\boldsymbol{\epsilon}$ such that $g^{\eta} \neq 1_{\mathbb{G}}$ and $e(g_p, g^{\eta}) = 1_{\mathbb{G}_T}$, $\mathcal{A}'$ outputs 1 if $e(g', g^{\eta}) = 1_{\mathbb{G}_T}$ and outputs 0 otherwise. Else, if there does not exist any such g^{η} in $\boldsymbol{\epsilon}$, $\mathcal{A}'$ outputs 1.

  When $g^{\eta} \in \mathbb{G}_q$ and $g^{\eta} \neq 1_{\mathbb{G}}$, we have $e(g', g^{\eta}) = 1_{\mathbb{G}_T}$ iff $g' \in \mathbb{G}_p$. This implies that when $e(g_p, g^{\eta}) = 1_{\mathbb{G}_T}$ and $g^{\eta} \neq 1_{\mathbb{G}}$ then $\mathcal{A}'$ solves the subgroup decision problem. It then follows from the subgroup decision assumption that such a g^{η} can be output by $\mathcal{A}$ with at most negligible probability which in turn implies from our discussion above that $\mathsf{Hyb}_0 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1$.
- Hyb_2: In this hybrid, the experiment always outputs 0.

  We show that $\mathsf{Hyb}_1 \overset{\mathsf{s}}{\approx} \mathsf{Hyb}_2$ by considering the following cases.

  - **Case 1:** There exists $k \in [1, \ell_{\mathsf{stmt}}]$ such that $g^{\epsilon_{u_k}} \neq 1_{\mathbb{G}}$.

    The statement consistency check always fails during verification if $g^{\epsilon_{u_k}} \neq 1_{\mathbb{G}}$ and so $\overline{\boldsymbol{\pi}}$ does not verify successfully and $\mathsf{Hyb}_1 \overset{\mathsf{p}}{\equiv} \mathsf{Hyb}_2$ in this case.
  - **Case 2:** $g^{\epsilon_{u_k}} = 1_{\mathbb{G}}$ for every $k \in [1, \ell_{\mathsf{stmt}}]$ but there exists $k \in [\ell_{\mathsf{stmt}} + 1, L]$ such that $g^{\epsilon_{u_k}} \neq 1_{\mathbb{G}}$ or there exists $k \in [1, L]$ such that $g^{\epsilon_{v_k}} \neq 1_{\mathbb{G}}$.

    We show that $\mathsf{Hyb}_1 \overset{\mathsf{s}}{\approx} \mathsf{Hyb}_2$ primarily because of the randomness used for zero-knowledge of the zkBARG. Note that if there exists $g^{\epsilon_{u_k}} \in \mathbb{G}_q$ or $g^{\epsilon_{v_k}} \in \mathbb{G}_q$ then both Hyb_1 and Hyb_2 output 0 in this case. Thus, we only consider the case when there exists $g^{\epsilon_{u_k}} \notin \mathbb{G}_q$ or $g^{\epsilon_{v_k}} \notin \mathbb{G}_q$.

    From the completeness of the zkBARG, for every $k \in [1, L]$ we have

    $$e(A, U_k) = e(g_p, V_k) \cdot e(U_k, U_k). \tag{3}$$

    Now, consider the k-th wire consistency check for $\overline{\boldsymbol{\pi}}$

    $$\begin{aligned} & e(A, \overline{U_k}) = e(g_p, \overline{V_k}) \cdot e(\overline{U_k}, \overline{U_k}) \\ \implies & e(A, U_k \cdot g^{\epsilon_{u_k}}) = e(g_p, V_k \cdot g^{\epsilon_{v_k}}) \cdot e(U_k \cdot g^{\epsilon_{u_k}}, U_k \cdot g^{\epsilon_{u_k}}) \\ \implies & e(A, g^{\epsilon_{u_k}}) = e(g_p, g^{\epsilon_{v_k}}) \cdot e(U_k, g^{\epsilon_{u_k}})^2 \cdot e(g^{\epsilon_{u_k}}, g^{\epsilon_{u_k}}) \end{aligned}$$

where the first implication follows from the definition of $\overline{\boldsymbol{\pi}}$ and the second implication follows from Equation (3). Observe that if $g^{\epsilon_{u_k}} = 1_{\mathbb{G}}$ and $g^{\epsilon_{v_k}} \notin \mathbb{G}_q$ then $e(g_p, g^{\epsilon_{v_k}}) \neq 1_{\mathbb{G}_T}$ and the k-th wire consistency check always fails for $\overline{\boldsymbol{\pi}}$. Moreover since $g^{\epsilon_{u_k}} = 1_{\mathbb{G}}$ for all $k \in [1, \ell_{\mathsf{stmt}}]$, if Hyb_1 is not equal to Hyb_2 then it must be the case that there exists $k \in [\ell_{\mathsf{stmt}} + 1, L]$ such that $g^{\epsilon_{u_k}} \notin \mathbb{G}_q$.

Next, we show that if there exists $g^{\epsilon_{u_k}} \notin \mathbb{G}_q$ for some $k \in [\ell_{\mathsf{stmt}} + 1, L]$ then $\overline{\boldsymbol{\pi}}$ verifies with at most negligible probability. If the k-th wire consistency check passes then $e(A, g^{\epsilon_{u_k}}) = e(g_p, g^{\epsilon_{v_k}}) \cdot e(U_k, g^{\epsilon_{u_k}})^2 \cdot e(g^{\epsilon_{u_k}}, g^{\epsilon_{u_k}})$, which along with the fact that $g_p = g^q$ and $\mathbb{G}_T$ is a cyclic group of order N with generator $g_T = e(g, g)$ implies that

$$q\epsilon_{u_k} \sum_{i=1}^{n} \alpha_i - q\epsilon_{v_k} - 2qu_k\epsilon_{u_k} \equiv \epsilon_{u_k}^2 \pmod{N}$$

where $U_k = g_p^{u_k}$. Furthermore, from the Chinese Remainder Theorem, $\epsilon_{u_k}^2 \equiv 0 \pmod{q}$, which in turn implies that $\epsilon_{u_k}^2 = q\nu$ for some $\nu \in \mathbb{N}$. Consequently,

$$\begin{aligned}
& q\epsilon_{u_k} \sum_{i=1}^{n} \alpha_i - q\epsilon_{v_k} - 2qu_k\epsilon_{u_k} - q\nu \equiv 0 \pmod{N} \\
\implies & q \cdot \left(\epsilon_{u_k} \sum_{i=1}^{n} \alpha_i - \epsilon_{v_k} - 2u_k\epsilon_{u_k} - \nu \right) \equiv 0 \pmod{N} \\
\implies & \epsilon_{u_k} \sum_{i=1}^{n} \alpha_i - \epsilon_{v_k} - 2u_k\epsilon_{u_k} - \nu \equiv 0 \pmod{p}. \qquad (4)
\end{aligned}$$

Since the adversary's view is independent of r_k and r_k is sampled uniformly at random from $\mathbb{Z}_N$ for every $k \in [\ell_{\mathsf{stmt}}+1, L]$, u_k is uniformly random over $\mathbb{Z}_N$ and independent of the error $\boldsymbol{\epsilon}$. However, since $g^{\epsilon_{u_k}} \notin \mathbb{G}_q$, we have $\epsilon_{u_k} \not\equiv 0 \pmod{p}$ and there exists at most one root for $\epsilon_{u_k} \sum_{i=1}^{n} \alpha_i - \epsilon_{v_k} - 2u_k\epsilon_{u_k} - \nu$ when viewed as a polynomial in u_k over $\mathbb{Z}_p$. From the Chinese Remainder Theorem there are q roots of this polynomial in $\mathbb{Z}_N$ which implies that Equation (4) holds with probability at most $q/N = 1/p \in \mathsf{negl}(\lambda)$. Therefore, Equation (4) holds with at most negligible probability when $g^{\epsilon_{u_k}} \notin \mathbb{G}_q$ which in turn implies that $\mathsf{zkBARG.Verify}(\mathsf{crs}, \mathbf{x}_1, \ldots, \mathbf{x}_n, \overline{\boldsymbol{\pi}}) = 0$ with overwhelming probability. Thus, it follows that when there exists $g^{\epsilon_{u_k}} \neq 1_{\mathbb{G}}$ or $g^{\epsilon_{v_k}} \neq 1_{\mathbb{G}}$ then $\mathsf{Hyb}_1 \stackrel{s}{\approx} \mathsf{Hyb}_2$.

– **Case 3:** $g^{\epsilon_{u_k}} = 1_{\mathbb{G}}$ for all $k \in [1, L]$, $g^{\epsilon_{v_k}} = 1_{\mathbb{G}}$ for all $k \in [1, L]$ but there exists $k \in [1, s]$ such that $g^{\epsilon_{w_k}} \neq 1_{\mathbb{G}}$.

If there exists $g^{\epsilon_{w_k}} \in \mathbb{G}_q$ then both Hyb_1 and Hyb_2 output 0. Thus, we only consider the case when there exists $g^{\epsilon_{w_k}} \notin \mathbb{G}_q$ which implies that $e(g_p, g^{\epsilon_{w_k}}) \neq 1_{\mathbb{G}_T}$. From the completeness of the zkBARG, the gate consistency checks for valid proofs always pass which implies that for the k-th NAND gate with input wires (a, b) and output wire c we have

$$e(A, A) \cdot e(g_p, W_k) = e(U_a, U_b) \cdot e(A, U_c). \qquad (5)$$

Now, consider the k-th gate consistency check for $\overline{\boldsymbol{\pi}}$

$$\begin{aligned}
& e(A, A) \cdot e(g_p, \overline{W_k}) = e(\overline{U_a}, \overline{U_b}) \cdot e(A, \overline{U_c}) \\
\implies & e(A, A) \cdot e(g_p, W_k \cdot g^{\epsilon_{w_k}}) = e(U_a \cdot g^{\epsilon_{u_a}}, U_b \cdot g^{\epsilon_{u_b}}) \cdot e(A, U_c \cdot g^{\epsilon_{u_c}}) \\
\implies & e(g_p, g^{\epsilon_{w_k}}) = e(U_a, g^{\epsilon_{u_b}}) \cdot e(g^{\epsilon_{u_a}}, U_b) \cdot e(g^{\epsilon_{u_a}}, g^{\epsilon_{u_b}}) \cdot e(A, g^{\epsilon_{u_c}})
\end{aligned}$$

where the first implication follows from the definition of $\overline{\boldsymbol{\pi}}$ and the second implication follow from Equation (5). However, since $g^{\epsilon_{u_k}} = 1_{\mathbb{G}}$ for every $k \in [1, L]$ in this case, it follows that

the k-th gate consistency check passes iff $e(g_p, g^{\epsilon_{w_k}}) = 1_{\mathbb{G}_T}$. Since there exists a $k \in [1, s]$ such that $e\left(g_p, g^{\epsilon_{w_k}}\right) \neq 1_{\mathbb{G}_T}$, it follows that at least one gate consistency check does not pass and that $\mathsf{zkBARG.Verify}(\mathsf{crs}, \mathbf{x}_1, \ldots, \mathbf{x}_n, \overline{\boldsymbol{\pi}}) = 0$ and $\mathsf{Hyb}_1 \stackrel{\mathrm{p}}{\equiv} \mathsf{Hyb}_2$ in this case.

It follows from our argument above that $\mathsf{Hyb}_1 \stackrel{\mathrm{s}}{\approx} \mathsf{Hyb}_2$.

Thus, $\mathsf{Hyb}_0 \stackrel{\mathrm{c}}{\approx} \mathsf{Hyb}_2$ from our sequence of hybrids and since Hyb_2 is always 0, it follows that the Waters-Wu zkBARG has robust verification. □

Observe that the zkBARG is succinct since the proof size is $O(\lambda(n + \ell_{\mathsf{wit}}))$ and the verifier runtime is $O(\lambda(n + \ell_{\mathsf{wit}} + n\ell_{\mathsf{stmt}}))$ while the length of the witness $(\mathbf{w}_1, \ldots, \mathbf{w}_n)$ is $O(n\ell_{\mathsf{wit}})$. Similarly, the zkBARG has succinct randomness since for a SIMD circuit of size ns it requires $O(s)$ uniformly random values from $\mathbb{Z}_N$ where each value is $\log \lambda$ bits long. It then follows from Theorem 10 and Claims 11.1 and 11.2 that the Waters-Wu zkBARG is a Splittable zkSNARG for SIMD boolean circuit satisfiability with non-adaptive soundness. ■

5 HSS with Verifiable Evaluation

In this section, we define HSS with verifiable evaluation and then construct it using HSS and Splittable zkSNARGs.

We define ve-HSS in Definition 7. As discussed in Section 1, a ve-HSS scheme allows a client to secret share a private input $\mathbf{x}$ among m servers. Given a public input $\mathbf{x}_{\mathsf{pub}}$, the servers can then locally compute on their shares to obtain shares of the output, which when reconstructed yields the circuit output $\mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$. However, since servers may be corrupt, they might send incorrect output shares. To address this, the scheme supports verification of the reconstructed candidate output $\mathbf{y}$. Specifically, we require two forms of verification: (1) *Local verification*, where the client, which knows the full input $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ can verify that $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$, and (2) *Public verification*, where any external entity can verify that the output is consistent with the public input i.e., there exists private input $\mathbf{x}$ such that $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$. We use π_{loc} and π_{pub} to denote the auxiliary output of reconstruction which is used for local and public verification, respectively.

Correctness requires that reconstruction indeed yields the circuit output when all servers are honest. Privacy guarantees that as long as at least one server is honest, the input shares of corrupt servers and the output of public verification preserve the privacy of the private input. Local (public) soundness ensures that local (public) verification fails if the candidate output is incorrect (inconsistent with the public input), even if all servers are corrupt. To ensure that the evaluation algorithm is non-trivial, we require that reconstruction is linear and that the runtime of the sharing algorithm is sublinear in the size of the circuit. To ensure that verification is succinct, we require that the output share size is sublinear in the size of the circuit and that the runtime of the local and public verification algorithms are sublinear in the size of the circuit and independent of the number of servers.

Definition 7 (HSS with verifiable evaluation)**.** *Let $m := m(\lambda)$ be an integer-valued polynomial, R be a ring and let $\mathbb{C}$ be a class of circuits over R. An m-server HSS with verifiable evaluation (ve-HSS) scheme for $\mathbb{C}$ is a tuple of algorithms* $\mathsf{ve\text{-}HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Recon}, \mathsf{LVerify}, \mathsf{PVerify})$ *with the following syntax.*

- $\mathsf{Setup}(1^\lambda, \mathsf{C}) \to \mathsf{crs}$ *is a* PPT *algorithm that takes the description of a circuit $\mathsf{C} \in \mathbb{C}$ as input and outputs the common reference string* crs*.*
- $\mathsf{Share}(\mathsf{crs}, \mathbf{x}) \to (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m)$ *is a* PPT *algorithm that takes as input the common reference string* crs *and private input $\mathbf{x} \in R^*$ and outputs a sharing of the inputs $\langle \mathbf{x} \rangle = (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m)$.*

ve-HSS Privacy Experiment

$\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$:

1: Compute $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$ and send crs to $\mathcal{A}$.

2: Receive $(\mathcal{I}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ from $\mathcal{A}$. Output 0 if $\mathcal{I} \not\subset [1, m]$ or if $|\mathcal{I}| = m$.

3: Compute

$$(\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m) \leftarrow \mathsf{Share}(\mathsf{crs}, \mathbf{x})$$

and send $\langle \mathbf{x} \rangle_{\mathcal{I}}$ to $\mathcal{A}$.

4: Receive $\left[\overline{\mathsf{out}}\right]_{\mathcal{I}}$ from $\mathcal{A}$. For each $i \in [1, m] \setminus \mathcal{I}$ compute

$$[\mathsf{out}]_i := \mathsf{Eval}(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i),$$

and then compute

$$(\overline{\mathbf{y}}, \overline{\pi_{\mathsf{loc}}}, \overline{\pi_{\mathsf{pub}}}) := \mathsf{Recon}(\mathsf{crs}, \left[\overline{\mathsf{out}}\right]_{\mathcal{I}}, [\mathsf{out}]_{[1,m]\setminus\mathcal{I}}),$$
$$b \leftarrow \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \overline{\mathbf{y}}, \overline{\pi_{\mathsf{loc}}})$$

5: Send $(b, \overline{\pi_{\mathsf{pub}}})$ to $\mathcal{A}$ and output b' returned by $\mathcal{A}$.

$\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$:

1: Compute $(\mathsf{crs}, \mathsf{st}) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{C})$ and send crs to $\mathcal{A}$.

2: Receive $(\mathcal{I}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ from $\mathcal{A}$. Output 0 if $\mathcal{I} \not\subset [1, m]$ or if $|\mathcal{I}| = m$.

3: Compute

$$(\langle \mathbf{x} \rangle_{\mathcal{I}}, \mathsf{st}) \leftarrow \mathsf{Sim}(\mathsf{st}, \mathcal{I}, |\mathbf{x}|)$$

and send $\langle \mathbf{x} \rangle_{\mathcal{I}}$ to $\mathcal{A}$.

4: Receive $\left[\overline{\mathsf{out}}\right]_{\mathcal{I}}$ from $\mathcal{A}$. Compute

$$(b, \overline{\pi_{\mathsf{pub}}}) \leftarrow \mathsf{Sim}\left(\mathsf{st}, \mathbf{x}_{\mathsf{pub}}, \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}), \left[\overline{\mathsf{out}}\right]_{\mathcal{I}}\right).$$

5: Send $(b, \overline{\pi_{\mathsf{pub}}})$ to $\mathcal{A}$ and output b' returned by $\mathcal{A}$.

Figure 5: ve-HSS privacy experiment for Definition 7.

- $\mathsf{Eval}(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i) =: [\mathsf{out}]_i$ *is a polynomial time algorithm that takes as input the common reference string* crs*, a server index* i*, the public input* $\mathbf{x}_{\mathsf{pub}} \in R^*$ *and the share of the* i*-th server. It outputs a share* $[\mathsf{out}]_i$*.*

- $\mathsf{Recon}(\mathsf{crs}, [\mathsf{out}]_1, \ldots, [\mathsf{out}]_m) =: (\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}})$ *is a polynomial time algorithm that takes as input the common reference string* crs *and the shares of the output, and reconstructs it to compute the circuit output* $\mathbf{y} \in R$ *and proofs* π_{loc} *and* π_{pub}*.*

- $\mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}}) \to b$ *is a* PPT *algorithm that takes the common reference string* crs*, the circuit inputs* $\mathbf{x}_{\mathsf{pub}}$ *and* $\mathbf{x}$*, the circuit output* $\mathbf{y}$ *and the local proof* π_{loc}*, and outputs a bit* b *indicating if the proof is accepted or rejected.*

- $\mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \to b$ *is a* PPT *algorithm that takes the common reference string* crs*, the public inputs* $\mathbf{x}_{\mathsf{pub}}$*, the circuit output* $\mathbf{y}$ *and the public proof* π_{pub}*, and outputs a bit* b *indicating if the proof is accepted or rejected.*

A ve-HSS scheme should satisfy the following properties.

- ***Efficiency:*** *The ve-HSS scheme is said to be efficient if it has linear reconstruction, succinct shares and succinct verification as defined below.*

- ***Linear Reconstruction:*** Recon *is linear in* $[\mathsf{out}]_1, \ldots, [\mathsf{out}]_m$.
- ***Succinct Shares:*** *There exists a polynomial* p *and a sublinear function* $g(x) \in o(x)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *with* ℓ_{out} *outputs, the runtime of* Share *when used for sharing an input of length* ℓ_{inp} *is at most* $p(m\lambda\ell_{\mathsf{inp}}) \cdot g(|\mathsf{C}|)$ *and the output length of* Eval *is at most* $p(\lambda\ell_{\mathsf{out}}) \cdot g(|\mathsf{C}|)$.
- ***Succinct Verification:*** *There exists a polynomial* p *and a sublinear function* $g(x) \in o(x)$ *such that for every* $\lambda, \ell_{\mathsf{inp}} \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *with* $(\ell_{\mathsf{inp}} + \ell_{\mathsf{pub}})$ *inputs and* ℓ_{out} *outputs, the runtime of* $\mathsf{LVerify}$ *is at most* $p\big(\lambda\big(\ell_{\mathsf{inp}} + \ell_{\mathsf{pub}} + \ell_{\mathsf{out}}\big)\big) \cdot g(|\mathsf{C}|)$ *and the runtime of* $\mathsf{PVerify}$ *is at most* $p\big(\lambda\big(\ell_{\mathsf{pub}} + \ell_{\mathsf{out}}\big)\big) \cdot g(|\mathsf{C}|)$. *Note that the verification time is* independent *of the number of servers* m.

- ***Correctness:*** *For all polynomial size adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *we have*

$$\Pr\left[\begin{array}{c} \mathbf{y} \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \vee \\ \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}}) = 0 \vee \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) = 0 \end{array} \middle| \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m) \leftarrow \mathsf{Share}(\mathsf{crs}, \mathbf{x}) \\ [\mathsf{out}]_i \coloneqq \mathsf{Eval}\big(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i\big),\ \forall i \in [1, m] \\ (\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}}) \coloneqq \mathsf{Recon}(\mathsf{crs}, [\mathsf{out}]_1, \ldots, [\mathsf{out}]_m) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

- ***Privacy:*** *There exists a* PPT *simulator* Sim *such that for all polynomial size adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$,

$$\left|\Pr\left[\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

  where $\mathsf{Privacy}^b_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$ *is defined in Figure 5.*

- ***Local Soundness:*** *For all polynomial size adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *we have*

$$\Pr\left[\begin{array}{c} \mathbf{y} \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \wedge \\ \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (\mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m) \leftarrow \mathsf{Share}(\mathsf{crs}, \mathbf{x}) \\ ([\mathsf{out}]_1, \ldots, [\mathsf{out}]_m) \leftarrow \mathcal{A}(\mathsf{st}, \langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m) \\ (\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}}) \coloneqq \mathsf{Recon}(\mathsf{crs}, [\mathsf{out}]_1, \ldots, [\mathsf{out}]_m) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

- ***Public Soundness:*** *For all polynomial size adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *we have*

$$\Pr\left[\begin{array}{c} (\mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}) \notin \mathsf{SAT} \wedge \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (\mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

We use $\langle \mathbf{x} \rangle = (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m)$ to denote the shares of all servers and use $\langle \mathbf{x} \rangle_{\mathcal{I}}$ as a shorthand to denote $\{\langle \mathbf{x} \rangle_i\}_{i \in \mathcal{I}}$ where $\mathcal{I} \subseteq [1, m]$.

Remarks. We next discuss some natural variants of our definition of ve-HSS that may be of interest.

- **Weak Privacy:** Definition 7 ensures a strong notion of privacy, which requires simulating the output of LVerify and the public proof π_{pub}, in addition to the shares of the inputs for corrupt parties. One could consider a weaker yet meaningful notion of privacy which only requires simulating the shares of inputs for corrupt parties, similar to the security of HSS (Definition 1).

– **Public Knowledge Soundness:** The public soundness property in Definition 7 can be strengthened by requiring knowledge soundness. Specifically, knowledge soundness requires that for all polynomial size adversaries $\mathcal{A}$ there exists a polynomial size extractor $\mathsf{Ext}_{\mathcal{A}}$ and a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}$,

$$\Pr\left[\begin{array}{c} \mathbf{y} \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \\ \wedge \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (\mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathbf{x} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

– **Non-adaptive Soundness And Privacy:** It is also meaningful to consider weaker, non-adaptive variants of the soundness and privacy definitions where the adversary first chooses the inputs and then receives the CRS. Specifically, a ve-HSS scheme is said to have non-adaptive public soundness if for all polynomial size adversaries there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}$ we have

$$\Pr\left[\begin{array}{c} (\mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}) \notin \mathsf{SAT} \\ \wedge \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) = 1 \end{array} \middle| \begin{array}{c} (\mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ \pi_{\mathsf{pub}} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{st}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

A similar definition follows for non-adaptive local soundness. The non-adaptive privacy experiment proceeds similarly to Figure 5 except that step 2 is run before step 1.

Construction. We present our construction of ve-HSS from HSS and Splittable in Figure 6. We refer the reader to Section 2.1 for an overview of the construction.

Theorem 12. *Let* HSS *be an m-server HSS scheme for a class of circuits* $\mathbb{C}_{\mathsf{hss}}$ *over a finite commutative ring* R *and let* SNARG *be a* $\mathbb{C}_{\mathsf{hss}}$*-simple Splittable zkSNARG for* $\mathbb{C}$*-circuit satisfiability such that* $\mathbb{C} \subseteq \mathbb{C}_{\mathsf{hss}}$*. Then, Figure 6 is an m-server ve-HSS scheme for* $\mathbb{C}$*. Moreover, it is black-box in* HSS *and the linear phase of* SNARG*.*

Proof. We argue that Figure 6 is a ve-HSS scheme by showing that it satisfies the required correctness, privacy, soundness and efficiency properties.

Correctness. For any $\mathsf{C} \in \mathbb{C}$, the correctness of HSS ensures that $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ with overwhelming probability. Additionally, the fact that SNARG is $\mathbb{C}_{\mathsf{hss}}$-simple implies that $[M_{\mathsf{loc}}]$ is an additive sharing of $M_{\mathsf{loc}} = \mathsf{SNARG.Prove}_{\mathsf{NL}}(\mathsf{C}_{\mathsf{loc}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}; r_{\mathsf{loc}})$ and $\left[M_{\mathsf{pub}}\right]$ is an additive sharing of $M_{\mathsf{pub}} = \mathsf{SNARG.Prove}_{\mathsf{L}}(\mathsf{C}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}; r_{\mathsf{pub}})$ with overwhelming probability. Finally, DP-completeness (Lemma 6) and the fact that SNARG is for $\mathbb{C}$-circuit satisfiability ensure that the reconstructed proofs π_{loc} and π_{pub} verify successfully with overwhelming probability.

Privacy. Intuitively, the security of HSS and zero-knowledge property of the publicly verifiable proof π_{pub} ensure the privacy of the input. Moreover, an adversary does not gain any advantage by deviating from the protocol and learning if local verification succeeded or failed due to DP-robust verification property of Splittable zkSNARGs. We formally show that the construction is private by constructing a simulator and arguing that any adversary $\mathcal{A}$ has only negligible advantage in winning the privacy experiment. Without loss of generality, we consider an adversary that corrupts all but one server.

Simulator:

1. It receives the circuit C as input and computes $\mathsf{crs}_{\mathsf{loc}}$ as in Setup and simulates $\mathsf{crs}_{\mathsf{pub}}$ using the SNARG simulator to compute $\mathsf{crs} = (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$. It outputs crs.

2. It then receives as input the corruption set $\mathcal{I}$ and length of the input ℓ_{inp}, and computes $\langle \mathbf{x} \rangle$, $\langle r_{\mathsf{loc}} \rangle$, and $\langle r_{\mathsf{pub}} \rangle$ as HSS shares of the zero vector of appropriate length. It outputs $\left(\langle \mathbf{x} \rangle_{\mathcal{I}}, \langle r_{\mathsf{loc}} \rangle_{\mathcal{I}}, \langle r_{\mathsf{pub}} \rangle_{\mathcal{I}}\right)$.

HSS with verifiable evaluation

Public parameters. Let HSS be an m-server HSS scheme that supports computation over a finite commutative ring R and let SNARG be a Splittable zkSNARG. Let ℓ_{rand} denote the number of random elements from R required by the zkSNARG prover.

Notation. For a circuit C, we use $\mathsf{C}_{\mathsf{loc}}$ and $\mathsf{C}_{\mathsf{pub}}$ to denote the circuit satisfiability instances for the local and public proof i.e., when $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ then $(\mathsf{C}_{\mathsf{loc}}, \mathbf{x}_{\mathsf{pub}} \| \mathbf{x}, \mathbf{y}) \in \mathsf{SAT}$ and $(\mathsf{C}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}) \in \mathsf{SAT}$. We use $\mathsf{C}^{\mathsf{loc}}_{\mathsf{NL}}$ and $\mathsf{C}^{\mathsf{pub}}_{\mathsf{NL}}$ to denote the circuits that compute the non-linear phase $\mathsf{Prove}_{\mathsf{NL}}$ of the Splittable zkSNARG for $\mathsf{C}_{\mathsf{loc}}$ and $\mathsf{C}_{\mathsf{pub}}$, respectively.

ve-HSS.Setup$(1^\lambda, \mathsf{C})$

1 : $\mathsf{crs}_{\mathsf{loc}} \leftarrow \mathsf{SNARG.Setup}(1^\lambda, \mathsf{C}_{\mathsf{loc}})$
2 : $\mathsf{crs}_{\mathsf{pub}} \leftarrow \mathsf{SNARG.Setup}(1^\lambda, \mathsf{C}_{\mathsf{pub}})$
3 : $\mathsf{crs} := (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
4 : **return** crs

ve-HSS.Share$(\mathbf{x})$

1 : $r_{\mathsf{loc}}, r_{\mathsf{pub}} \leftarrow R^{\ell_{\mathsf{rand}}}$
2 : $\langle \mathbf{x} \rangle, \langle r_{\mathsf{loc}} \rangle, \langle r_{\mathsf{pub}} \rangle \leftarrow \mathsf{HSS.Share}(1^\lambda, \mathbf{x}, r_{\mathsf{loc}}, r_{\mathsf{pub}})$
3 : **return** $\left(\langle \mathbf{x} \rangle, \langle r_{\mathsf{loc}} \rangle, \langle r_{\mathsf{pub}} \rangle\right)$

ve-HSS.Eval$(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{loc}} \rangle_i, \langle r_{\mathsf{pub}} \rangle_i)$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
2 : $[\mathbf{y}]_i := \mathsf{HSS.Eval}\left(i, \mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i\right)$
3 : $[M_{\mathsf{loc}}]_i := \mathsf{HSS.Eval}\left(i, \mathsf{C}^{\mathsf{loc}}_{\mathsf{NL}}, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{loc}} \rangle_i\right)$
4 : $[\pi_{\mathsf{loc}}]_i := \mathsf{SNARG.Prove}_{\mathsf{L}}(\mathsf{crs}_{\mathsf{loc}}, [M_{\mathsf{loc}}]_i)$
5 : $\left[M_{\mathsf{pub}}\right]_i := \mathsf{HSS.Eval}\left(i, \mathsf{C}^{\mathsf{pub}}_{\mathsf{NL}}, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{pub}} \rangle_i\right)$
6 : $\left[\pi_{\mathsf{pub}}\right]_i := \mathsf{SNARG.Prove}_{\mathsf{L}}\left(\mathsf{crs}_{\mathsf{pub}}, \left[M_{\mathsf{pub}}\right]_i\right)$
7 : $[\mathsf{out}]_i = \left([\mathbf{y}]_i, [\pi_{\mathsf{loc}}]_i, \left[\pi_{\mathsf{pub}}\right]_i\right)$
8 : **return** $[\mathsf{out}]_i$

ve-HSS.Recon$([\mathsf{out}]_1, \ldots, [\mathsf{out}]_m)$

1 : **parse each** $[\mathsf{out}]_i = \left([\mathbf{y}]_i, [\pi_{\mathsf{loc}}]_i, \left[\pi_{\mathsf{pub}}\right]_i\right)$
2 : $\mathbf{y} := \sum_{i=1}^{m} [\mathbf{y}]_i$
3 : $\pi_{\mathsf{loc}} := \sum_{i=1}^{m} [\pi_{\mathsf{loc}}]_i$
4 : $\pi_{\mathsf{pub}} := \sum_{i=1}^{m} \left[\pi_{\mathsf{pub}}\right]_i$
5 : $\mathsf{out} := (\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}})$
6 : **return** out

ve-HSS.LVerify$(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}})$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
2 : $b := \mathsf{SNARG.Verify}(\mathsf{crs}_{\mathsf{loc}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}})$
3 : **return** b

ve-HSS.PVerify$(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}})$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
2 : $b := \mathsf{SNARG.Verify}(\mathsf{crs}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}})$
3 : **return** b

Figure 6: ve-HSS construction from HSS and Splittable zkSNARG.

3. It then receives the public input $\mathbf{x}_{\mathsf{pub}}$, the circuit output $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ and the evaluated shares of the corrupt servers $\left[\overline{\mathsf{out}}\right]_{\mathcal{I}} = \left([\overline{\mathbf{y}}]_{\mathcal{I}}, [\overline{\pi_{\mathsf{loc}}}]_{\mathcal{I}}, \left[\overline{\pi_{\mathsf{pub}}}\right]_{\mathcal{I}}\right)$ as input and does the following.

   (a) For every $i \in \mathcal{I}$, it re-computes the output share of the i-th server as

$$\left([\mathbf{y}]_i, [\pi_{\mathsf{loc}}]_i, \left[\pi_{\mathsf{pub}}\right]_i\right) \coloneqq \mathsf{Eval}\left(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{loc}} \rangle_i, \langle r_{\mathsf{pub}} \rangle_i\right).$$

   (b) It computes

$$\epsilon_y = \sum_{i \in \mathcal{I}} [\overline{\mathbf{y}}]_i - [\mathbf{y}]_i, \qquad \epsilon_{\mathsf{loc}} = \sum_{i \in \mathcal{I}} [\overline{\pi_{\mathsf{loc}}}]_i - [\pi_{\mathsf{loc}}]_i, \qquad \epsilon_{\mathsf{pub}} = \sum_{i \in \mathcal{I}} \left[\overline{\pi_{\mathsf{pub}}}\right]_i - \left[\pi_{\mathsf{pub}}\right]_i.$$

   (c) It simulates $\pi_{\mathsf{pub}} \leftarrow \mathsf{SNARG.Sim}(\mathsf{st}_s, \mathbf{x}_{\mathsf{pub}}, \mathbf{y})$ and computes $\overline{\pi_{\mathsf{pub}}} \coloneqq \pi_{\mathsf{pub}} + \epsilon_{\mathsf{pub}}$.

   (d) If $\epsilon_y \neq 0$ or if $\epsilon_{\mathsf{loc}} \neq 0$ then it sets $b \coloneqq 0$; else it sets $b \coloneqq 1$.

   (e) It outputs $(b, \overline{\pi_{\mathsf{pub}}})$.

Consider any polynomial size adversary $\mathcal{A}$ for the Privacy experiment. We show using a hybrid argument that for any circuit $\mathsf{C} \in \mathbb{C}$, the view of $\mathcal{A}$ in $\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}$ is indistinguishable from that in $\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}$.

- Hyb_0: This corresponds to the output of $\mathcal{A}$ in $\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$.
- Hyb_1: In this hybrid, the experiment runs step 4 in the following manner.
  - For each $i \in \mathcal{I}$, it computes the output share $[\mathsf{out}]_i$ of the corrupt server as

$$\left([\mathbf{y}]_i, [\pi_{\mathsf{loc}}]_i, \left[\pi_{\mathsf{pub}}\right]_i\right) \coloneqq \mathsf{Eval}\left(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{loc}} \rangle_i, \langle r_{\mathsf{pub}} \rangle_i\right).$$

  - It computes the reconstructed output $\mathsf{out} = (\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}})$ as

$$\begin{aligned} \mathbf{y} &\coloneqq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}), \\ \pi_{\mathsf{loc}} &\coloneqq \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}_{\mathsf{loc}}, \mathsf{Prove}_{\mathsf{NL}}(\mathsf{C}_{\mathsf{loc}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}; r_{\mathsf{loc}})), \\ \pi_{\mathsf{pub}} &\coloneqq \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}_{\mathsf{pub}}, \mathsf{Prove}_{\mathsf{NL}}(\mathsf{C}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}; r_{\mathsf{pub}})). \end{aligned}$$

  - It then computes the share of the honest server as

$$[\mathbf{y}]_{i^*} = \mathbf{y} - \sum_{i \in \mathcal{I}} [\mathbf{y}]_i, \qquad [\pi_{\mathsf{loc}}]_{i^*} = \pi_{\mathsf{loc}} - \sum_{i \in \mathcal{I}} [\pi_{\mathsf{loc}}]_i, \qquad \left[\pi_{\mathsf{pub}}\right]_{i^*} = \pi_{\mathsf{pub}} - \sum_{i \in \mathcal{I}} \left[\pi_{\mathsf{pub}}\right]_i.$$

  - It then continues to run step 4 of the $\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}$ experiment using $[\mathsf{out}]_{i^*}$ and $\left[\overline{\mathsf{out}}\right]_{\mathcal{I}}$, where the latter is the output shares returned by the adversary.

In this hybrid, we essentially simulate the output share of the honest server as discussed in Remark 1. The main difference between Hyb_0 and Hyb_1 is that in Hyb_0 the output shares of the honest servers were computed using Eval while in this hybrid, they are reverse sampled using the circuit output and proofs and the corrupt servers' shares of the circuit output and proofs. Specifically, since $\mathsf{C} \in \mathbb{C}_{\mathsf{hss}}$ and SNARG is $\mathbb{C}_{\mathsf{hss}}$-simple, it follows from the correctness of HSS and the linearity of additive secret sharing that the output shares of honest servers computed using Eval in Hyb_0 indeed correspond to shares of $\mathbf{y}$, π_{loc}, and π_{pub} as computed in Hyb_1. Moreover, since $\mathcal{I} \subset [1, m]$, the privacy of additive secret sharing imply that the shares of honest servers reverse sampled in Hyb_1 correspond to shares of $\mathbf{y}$, π_{loc}, and π_{pub}. Thus, it follows that $\mathsf{Hyb}_0 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1$.

– Hyb_2 : In this hybrid, the experiment sends the HSS shares of the input computed by the simulator to the adversary.

The only difference in this hybrid is that $\langle \mathbf{x} \rangle_{\mathcal{I}}$, $\langle r_{\mathsf{loc}} \rangle_{\mathcal{I}}$, and $\langle r_{\mathsf{pub}} \rangle_{\mathcal{I}}$ sent to the adversary are now computed as shares of 0 instead of the circuit inputs. Observe that this does not impact the other steps of the experiment since the experiment no longer uses the HSS shares of honest servers. It follows that $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$ from the security of HSS.

– Hyb_3: In this hybrid, the experiment outputs $b = 0$ if $\epsilon_y \neq 0$ where ϵ_y is as computed by the simulator.

Observe that

$$\overline{\mathbf{y}} = [\mathbf{y}]_{i^*} + \sum_{i \in \mathcal{I}} [\overline{\mathbf{y}}]_i = \mathbf{y} + \sum_{i \in \mathcal{I}} [\overline{\mathbf{y}}]_i - [\mathbf{y}]_i = \mathbf{y} + \epsilon_y$$

where the first equality follows from the definition of $\overline{\mathbf{y}}$ and the third equality follows from the definition of ϵ_y. Thus, if $\epsilon_y \neq 0$, then $\overline{\mathbf{y}} \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ which then implies that LVerify outputs 0 with overwhelming probability in Hyb_2 from the soundness of the SNARG and the fact that $\mathsf{C} \in \mathbb{C}$. It follows that $\mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3$.

– Hyb_4: In this hybrid, the experiment additionally outputs $b = 0$ when $\epsilon_{\mathsf{loc}} \neq 0$.

Observe that $\mathcal{A}$'s view is independent of r_{loc} used to compute π_{loc} since the HSS shares are simulated. It follows from the DP-robust verification property of SNARG (Lemma 7) and the fact that SNARG is a zkSNARG for $\mathbb{C}$-circuit satisfiability that LVerify outputs 0 in Hyb_3 with overwhelming probability when $\epsilon_{\mathsf{loc}} \neq 0$. Thus, $\mathsf{Hyb}_3 \overset{c}{\approx} \mathsf{Hyb}_4$.

– Hyb_5 : In this hybrid, the experiment outputs $b = 1$ when $\epsilon_y = 0$ and $\epsilon_{\mathsf{loc}} = 0$.

When $\epsilon_y = 0$ and $\epsilon_{\mathsf{loc}} = 0$, we have $\overline{\mathbf{y}} = \mathbf{y}$ and $\overline{\pi_{\mathsf{loc}}} = \pi_{\mathsf{loc}}$ respectively. Since $\mathsf{C} \in \mathbb{C}$, it follows from the DP-completeness (Lemma 6) of SNARG that that $\mathsf{LVerify}(\mathsf{crs}_{\mathsf{loc}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \overline{\mathbf{y}}, \overline{\pi_{\mathsf{loc}}}) = 1$ in Hyb_4 with overwhelming probability. Thus, $\mathsf{Hyb}_4 \overset{c}{\approx} \mathsf{Hyb}_5$.

– Hyb_6: In this hybrid, the experiment computes $\mathsf{crs}_{\mathsf{pub}}$ and $\overline{\pi_{\mathsf{pub}}}$ using the simulator.

Observe that

$$\overline{\pi_{\mathsf{pub}}} = \left[\pi_{\mathsf{pub}}\right]_{i^*} + \sum_{i \in \mathcal{I}} \left[\overline{\pi_{\mathsf{pub}}}\right]_i = \pi_{\mathsf{pub}} + \sum_{i \in \mathcal{I}} \left[\overline{\pi_{\mathsf{pub}}}\right]_i - \left[\pi_{\mathsf{pub}}\right]_i = \pi_{\mathsf{pub}} + \epsilon_{\mathsf{pub}}$$

where the first equality follows from the definition of $\overline{\pi_{\mathsf{pub}}}$ and the third equality follows from the definition of ϵ_{pub}. Moreover, the adversary's view is independent of the randomness r_{pub} used to compute π_{pub} in Hyb_5 since the HSS shares $\langle r_{\mathsf{pub}} \rangle_{\mathcal{I}}$ are simulated, which from the zero-knowledge property of SNARG implies that $\mathsf{Hyb}_5 \overset{c}{\approx} \mathsf{Hyb}_6$.

Hyb_6 corresponds to the output of $\mathcal{A}$ in $\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$ which in turn implies from our sequence of hybrids that

$$\left|\Pr\left[\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Soundness and efficiency. Local and public soundness of Figure 6 follow immediately from the soundness of SNARG. Moreover, the construction has succinct input shares since SNARG has succinct randomness and it has succinct output shares since HSS is compact and SNARG has succinct proof size. Succinct verification of the ve-HSS construction follows from succinct verification of SNARG. It then follows from our argument above that Figure 6 is an m-server ve-HSS scheme for $\mathbb{C}$. ■

Public knowledge soundness. Observe that public soundness of Figure 6 follows directly from the soundness of SNARG. Thus, if SNARG is a Splittable zkSNARK then the resulting ve-HSS scheme additionally has public knowledge soundness.

Remark 5 (On Requiring HSS with Negligible Correctness Error)**.** *Note that privacy of the ve-HSS construction (specifically, the indistinguishability of* Hyb_0 *and* Hyb_1*) requires that the underlying HSS scheme have negligible correctness error. As a result, HSS schemes such as that of [BGI16a], which have inverse polynomial correctness error—where the probability of error depends on the input—not only induce an inverse polynomial correctness error in the resulting ve-HSS from Figure 6, but also weaken the privacy guarantee. In particular, learning if the output reconstruction was successful could leak information about the private inputs. However, the resulting ve-HSS construction still satisfies weak privacy as defined above, which guarantees that the adversary's input shares do not provide any additional information about the inputs.*

Remark 6 (Extension to threshold HSS)**.** *In this work, we primarily consider m-out-of-m HSS schemes, where reconstruction requires the output shares of all m servers and privacy is guaranteed as long as at least one server is honest. However, we note that our techniques naturally extend to HSS schemes with arbitrary access structures. Specifically, our construction only requires that the output shares of HSS correspond to that of a linear secret sharing scheme and that the output shares of honest servers are efficiently sampleable given the output shares of corrupt servers.*

5.1 Instantiations

We now discuss instantiating our ve-HSS template described in Figure 6 using HSS schemes from prior works and the Splittable zkSNARGs discussed in Section 4, namely Groth16 and the Waters-Wu zkBARG.

In the remainder of this section we will use $\mathsf{VP}[p]$ to denote the set of polynomial size circuits over $\mathbb{Z}_p$ and use $\mathsf{VNC}^1[p]$ denote the set of polynomial size circuits over $\mathbb{Z}_p$ that have logarithmic depth in the number of inputs.

Recall that our ve-HSS construction requires an HSS scheme with negligible correctness error that supports evaluating the non-linear phase of the Splittable zkSNARG. As discussed in Section 3.1.1, there are two broad groups of HSS schemes with negligible correctness error in the literature. The first group consists of 2-server HSS schemes from group-based assumptions [BKS19, OSY21, RS21, ADOS22] that support evaluating RMS programs with an exponentially large magnitude bound. From Lemma 5, these HSS schemes can also be used to evaluate circuits in $\mathsf{VNC}^1[p]$ for $p \in 2^{O(\lambda)}$.[17] The second group consists of m-server HSS schemes for evaluating $\mathsf{VP}[p]$ [BGI15, COS+22], where m is polynomial in λ and $p \in 2^{O(\lambda)}$. Thus, our discussion below will focus on two classes of ve-HSS instantiations, depending on the underlying HSS scheme.

Groth16 based ve-HSS. From Theorem 9, we know that the non-linear phase of Groth16, when proving satisfiability of a boolean circuit of depth D, requires evaluating a circuit of depth $O(D)$ over $\mathbb{Z}_p$, where $p \in O(2^\lambda)$ is the order of the bilinear group used for Groth16. It then follows that Groth16 is a $\mathsf{VNC}^1[p]$-simple Splittable zkSNARG for NC^1-circuit satisfiability and that it is a $\mathsf{VP}[p]$-simple Splittable zkSNARG for $\mathsf{P/poly}$-circuit satisfiability. The following corollary of Theorem 12 then follows from the fact that HSS schemes from prior works support computing over $\mathsf{VNC}^1[p]$ and $\mathsf{VP}[p]$, as discussed above.

Corollary 13. *There exist the following ve-HSS schemes with local and public knowledge soundness in the generic bilinear group model:*

[17] In general, the RMS programs resulting from Lemma 5 will have a magnitude bound of $2^{O(\lambda \cdot n)}$, leading to HSS input shares of size $\mathsf{poly}(\lambda \cdot n^2)$ bits in existing group-based schemes. However, as noted in Remark 7, our ve-HSS instantiations only require bounds of $2^{O(\lambda)}$, reducing the share size to $O(\lambda \cdot n)$ bits.

- *A 2-server ve-HSS scheme for* NC^1 *assuming the hardness of the LWE assumption,*
- *A 2-server ve-HSS scheme for* NC^1 *assuming the hardness of the DCR assumption,*
- *A 2-server ve-HSS scheme for* NC^1 *assuming the hardness of the DDH assumption and the small exponent assumption over class groups,*
- *An* $m(\lambda)$*-server ve-HSS scheme for* $\mathsf{P/poly}$*, for any integer valued polynomial* $m(\cdot)$*, assuming the existence of sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure one-way functions,*
- *An* $m(\lambda)$*-server ve-HSS scheme for* $\mathsf{P/poly}$*, for any integer valued polynomial* $m(\cdot)$*, assuming the existence of circular secure AFS-spooky encryption.*

We note that verifiable evaluation comes at little to no cost to the input and output share sizes in this case. Specifically, the Groth16 prover requires two uniformly random elements from $\mathbb{Z}_p$ and the Groth16 proof consists of 3 group elements each of which is $O(\log \lambda)$ bits long. Thus, Figure 6 when instantiated with an HSS scheme HSS and Groth16 incurs only an additive overhead of $O(\log \lambda)$ to the size of the input and output shares compared to that of HSS.

Waters-Wu zkBARG based ve-HSS. Similar to Groth16, it follows from Theorem 11 that the Waters-Wu zkBARG is a $\mathsf{VNC}^1[N]$-simple zkSNARG for SIMD NC^1-circuit satisfiability and is a $\mathsf{VP}[N]$-simple zkSNARG for SIMD $\mathsf{P/poly}$-circuit satisfiability, where N is the order of the bilinear groups used for the zkBARG. Since the zkBARG is only non-adaptively sound, the resulting ve-HSS only achieves non-adaptive local and public soundness and non-adaptive privacy.[18] Thus, we have the following corollary of Theorem 12.

Corollary 14. *There exist the following ve-HSS schemes with non-adaptive local and public soundness, and non-adaptive privacy, all of which assume the hardness of the subgroup decision assumption over composite-order bilinear groups:*

- *A 2-server ve-HSS scheme for SIMD* NC^1 *additionally assuming the hardness of the LWE assumption,*
- *A 2-server ve-HSS scheme for SIMD* NC^1 *additionally assuming the hardness of the DCR assumption,*
- *A 2-server ve-HSS scheme for SIMD* NC^1 *additionally assuming the hardness of the DDH assumption and the small exponent assumption over class groups,*
- *An* $m(\lambda)$*-server ve-HSS scheme for SIMD* $\mathsf{P/poly}$*, for any integer valued polynomial* $m(\cdot)$*, additionally assuming the existence of sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure one-way functions,*
- *An* $m(\lambda)$*-server ve-HSS scheme for SIMD* $\mathsf{P/poly}$*, for any integer valued polynomial* $m(\cdot)$*, additionally assuming the existence of circular secure AFS-spooky encryption.*

Remark 7. *As discussed earlier, existing group-based HSS schemes support evaluating RMS programs over bounded integers, which from Lemma 5, suffice for the non-linear phases of Groth16 and the Waters-Wu zkBARG when proving satisfiability of* NC^1 *and SIMD* NC^1 *circuits, respectively. When these non-linear phases are evaluated using RMS programs, the values are bounded by* $O(p^2)$ *for Groth16 and* $O(N^2)$ *for the Waters-Wu zkBARG. This follows from the fact that a magnitude bound of 1 suffices to compute the extended witness*

[18] Non-adaptive privacy stems from using the soundness of the zkBARG to ensure correctness of the output in the security proof. Standard privacy can be recovered using a MAC check, as done in the multi-client construction in Section 6.

produced by the non-linear phase—since these are just intermediate values in a Boolean circuit [BGI16a]—and the only additional operations involve pairwise multiplications of random values in $\mathbb{Z}_p$ (or $\mathbb{Z}_N$ for Waters-Wu) and their products with the extended witness. Hence, all intermediate values remain bounded by $O(p^2)$ and $O(N^2)$ in the respective constructions.

Finally, we note that an HSS scheme supporting RMS programs with magnitude bounds polynomial in the security parameter also suffices to instantiate the ve-HSS scheme. Specifically, the extended witness from the non-linear phase can be computed using such a scheme, as discussed above. Since the computation over the larger ring has constant depth, it can be expressed by a log-depth Boolean circuit [vzGS91] and thus by an RMS program with magnitude bound 1 [BGI16a]. However, this requires sharing each bit of the non-binary ring elements and involves non-black-box use of arithmetic over the large ring.

6 Multi-Client HSS with Verifiable Evaluation

In this section, we generalize our ve-HSS definition to accommodate multiple clients. We first define multi-client ve-HSS for semi-honest clients and malicious servers in Definition 8, then strengthen the definition to model maliciously corrupt clients and servers in Definition 9.

Similar to public-key HSS (Definition 2), a multi-client ve-HSS scheme generates a common public key pk and an evaluation key ek_i for each server. Each client P_i^c computes an encryption $\widetilde{\mathbf{x}}_i$ of its input $\mathbf{x}_i$ using the public key, which it sends to all servers. Given a public input $\mathbf{x}_{\mathsf{pub}}$ and the client ciphertexts $\{\widetilde{\mathbf{x}}_i\}_i$, each server uses its evaluation key to locally compute a share of the output $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$. Public verification supports verifying that the reconstructed output $\mathbf{y}$ is consistent with the public input $\mathbf{x}_{\mathsf{pub}}$. Moreover, each server obtains an output share for each client P_i^c, which when reconstructed, allows P_i^c to locally verify if the output $\mathbf{y}$ is consistent with its input $\mathbf{x}_i$ i.e., there exists $\{\mathbf{x}_j\}_{j\neq i}$ such that $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$. Privacy requires that an adversary who semi-honestly corrupts a subset of the clients and maliciously corrupts a subset of the servers learns nothing beyond the inputs of the corrupt clients and the output, even if it learns whether verification by honest clients succeeds or fails. Similarly, correctness, efficiency, and soundness are straightforward extensions of the properties defined for ve-HSS.

Definition 8 (Multi-Client HSS with verifiable evaluation)**.** *Let $m := m(\lambda)$ and $n := n(\lambda)$ be integer-valued polynomials, R be a ring, and $\mathbb{C}$ be a class of circuits over R. An n-client, m-server Multi-Client HSS with verifiable evaluation (MveHSS) scheme for $\mathbb{C}$ is a tuple of algorithms* $\mathsf{MveHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Recon}, \mathsf{LVerify}, \mathsf{PVerify})$ *with the following syntax.*

- $\mathsf{Setup}(1^\lambda, \mathsf{C}) \to \mathsf{crs}$ *is a* PPT *algorithm that takes the description of a circuit $\mathsf{C} \in \mathbb{C}$ as input and outputs the common reference string* crs*.*
- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m)$ *is a* PPT *algorithm that outputs a public key* pk *and an evaluation key ek_i for the i-th server, for each $i \in [1, m]$.*
- $\mathsf{Enc}(\mathsf{crs}, i, \mathsf{pk}, \mathbf{x}) \to (\mathsf{st}, \widetilde{\mathbf{x}})$ *is a* PPT *algorithm that takes as input the common reference string* crs*, the client index i, the public key* pk*, the client input $\mathbf{x}$ and outputs a ciphertext $\widetilde{\mathbf{x}}$ and local client state* st*.*
- $\mathsf{Eval}(\mathsf{crs}, \mathsf{ek}_i, \mathbf{x}_{\mathsf{pub}}, \widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_n) =: \left(\left[\mathsf{out}_{\mathsf{pub}}\right]_i, [\mathsf{out}_1]_i, \ldots, [\mathsf{out}_n]_i\right)$ *is a polynomial time algorithm that takes as input the common reference string* crs*, the evaluation key of the i-th server ek_i, the public input $\mathbf{x}_{\mathsf{pub}}$, and the ciphertexts corresponding to the inputs of n clients. It outputs a share of the j-th client's output $\left[\mathsf{out}_j\right]_i$, for each $j \in [1, n]$, as well as a share of the public output $\left[\mathsf{out}_{\mathsf{pub}}\right]_i$.*
- $\mathsf{Recon}(\mathsf{crs}, [\mathsf{out}_i]) =: (\mathbf{y}, \pi_{\mathsf{pub}})/\pi_i$ *is a polynomial time algorithm that takes the shares of the public output $\left[\mathsf{out}_{\mathsf{pub}}\right]$ and reconstructs it to compute the circuit output $\mathbf{y}$ and public proof π_{pub}. When run on the shares of the i-th client's output $[\mathsf{out}_i]$, it outputs the proof π_i for the i-th client.*

MveHSS Privacy Experiment

$\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$

1: Compute $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$ and send crs to $\mathcal{A}$.

2: Receive $(\mathcal{I}, \mathcal{I}_\mathsf{c}, \mathbf{x}_\mathsf{pub}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$ from $\mathcal{A}$. Output 0 if $\mathcal{I}_\mathsf{c} \not\subset [1,n]$, $\mathcal{I} \not\subset [1,m]$ or if $|\mathcal{I}| = m$.

3: Compute

$$(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{KeyGen}(1^\lambda)$$
$$\forall i \in [1,n],\ \left(\mathsf{st}_i, \widetilde{\mathbf{x}}_i\right) \leftarrow \mathsf{Enc}(\mathsf{crs}, i, \mathsf{pk}, \mathbf{x}_i)$$

and send pk, $\{\mathsf{ek}_i\}_{i\in\mathcal{I}}$, $\left\{\mathsf{st}_i\right\}_{i\in\mathcal{I}_\mathsf{c}}$, and $(\widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_n)$ to $\mathcal{A}$.

4: Receive $\left[\overline{\mathsf{out}_\mathsf{pub}}\right]_\mathcal{I}, \left[\overline{\mathsf{out}_1}\right]_\mathcal{I}, \ldots, \left[\overline{\mathsf{out}_n}\right]_\mathcal{I}$ from $\mathcal{A}$.

5: For each $i \in [1,m] \setminus \mathcal{I}$, compute

$$\left(\left[\mathsf{out}_\mathsf{pub}\right]_i, \left[\mathsf{out}_1\right]_i, \ldots, \left[\mathsf{out}_n\right]_i\right) := \mathsf{Eval}(\mathsf{crs}, \mathsf{ek}_i, \mathbf{x}_\mathsf{pub}, \widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_n).$$

Then, compute

$$(\overline{\mathbf{y}}, \overline{\pi_\mathsf{pub}}) := \mathsf{Recon}\left(\mathsf{crs}, \left[\overline{\mathsf{out}_\mathsf{pub}}\right]_\mathcal{I}, \left[\mathsf{out}_\mathsf{pub}\right]_{[1,m]\setminus\mathcal{I}}\right),$$

and for each $i \in [1,n]$ compute

$$\overline{\pi_i} := \mathsf{Recon}\left(\mathsf{crs}, \left[\overline{\mathsf{out}_i}\right]_\mathcal{I}, \left[\mathsf{out}_i\right]_{[1,m]\setminus\mathcal{I}}\right)$$
$$b_i \leftarrow \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_\mathsf{pub}, \mathbf{x}_i, \overline{\mathbf{y}}, \mathsf{st}_i, \overline{\pi_i}).$$

6: Send $\left(\{b_i\}_{i\notin\mathcal{I}_\mathsf{c}}, \overline{\pi_\mathsf{pub}}, \{\overline{\pi}_i\}_{i\in\mathcal{I}_\mathsf{c}}\right)$ to $\mathcal{A}$ and output the bit returned by $\mathcal{A}$.

$\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$

1: Compute $(\mathsf{crs}, \mathsf{st}) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{C})$ and send crs to $\mathcal{A}$.

2: Receive $(\mathcal{I}, \mathcal{I}_\mathsf{c}, \mathbf{x}_\mathsf{pub}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$ from $\mathcal{A}$. Output 0 if $\mathcal{I}_\mathsf{c} \notin [1,n]$, $\mathcal{I} \not\subset [1,m]$ or if $|\mathcal{I}| = m$.

3: Compute

$$\left(\mathsf{pk}, \{\mathsf{ek}_i\}_{i\in\mathcal{I}}, \left\{\mathsf{st}_i\right\}_{i\in\mathcal{I}_\mathsf{c}}, \{\widetilde{\mathbf{x}}_i\}_{i=1}^n, \mathsf{st}\right) \leftarrow \mathsf{Sim}\left(\mathsf{st}, \mathsf{pk}, \{|\mathbf{x}_i|\}_{i\notin\mathcal{I}_\mathsf{c}}, \{\mathbf{x}_i\}_{i\in\mathcal{I}_\mathsf{c}}\right)$$

and send pk, $\{\mathsf{ek}_i\}_{i\in\mathcal{I}}$ and $(\widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_n)$ to $\mathcal{A}$.

4: Receive $\left[\overline{\mathsf{out}_\mathsf{pub}}\right]_\mathcal{I}, \left[\overline{\mathsf{out}_1}\right]_\mathcal{I}, \ldots, \left[\overline{\mathsf{out}_n}\right]_\mathcal{I}$ from $\mathcal{A}$.

5: Compute

$$\left(\{b_i\}_{i\notin\mathcal{I}_\mathsf{c}}, \overline{\pi_\mathsf{pub}}, \{\overline{\pi}_i\}_{i\in\mathcal{I}_\mathsf{c}}\right) \leftarrow \mathsf{Sim}\left(\mathsf{st}, \mathbf{x}_\mathsf{pub}, \mathbf{y}, \left[\overline{\mathsf{out}_\mathsf{pub}}\right]_\mathcal{I}, \left\{\left[\overline{\mathsf{out}_i}\right]_\mathcal{I}\right\}_{i=1}^n\right)$$

where $\mathbf{y} = \mathsf{C}(\mathbf{x}_\mathsf{pub}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$.

6: Send $\left(\{b_i\}_{i\notin\mathcal{I}_\mathsf{c}}, \overline{\pi_\mathsf{pub}}, \{\overline{\pi}_i\}_{i\in\mathcal{I}_\mathsf{c}}\right)$ to $\mathcal{A}$ and output the bit returned by $\mathcal{A}$.

Figure 7: MveHSS privacy experiment for Definition 8.

- $\mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_i, \mathbf{y}, \mathsf{st}_i, \pi_i) \to b$ *is a* PPT *algorithm that takes the common reference string* crs*, the public input* $\mathbf{x}_{\mathsf{pub}}$*, the* i*-th client's input* $\mathbf{x}_i$ *and local state* st_i*, the circuit output* $\mathbf{y}$*, and a proof* π_i *as input. It outputs a bit* b *indicating if* π_i *is accepted or rejected.*
- $\mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \to b$ *is a* PPT *algorithm that takes the common reference string* crs*, the public input* $\mathbf{x}_{\mathsf{pub}}$*, the circuit output* $\mathbf{y}$ *and a proof* π_{pub} *as input, and outputs a bit* b *indicating if* π_{pub} *is accepted or rejected.*

An MveHSS scheme should satisfy the following properties.

- ***Efficiency:*** *An MveHSS scheme is said to be efficient if it has linear reconstruction, succinct shares, succinct verification, and circuit independent key generation as defined below.*
  - ***Linear Reconstruction:*** $\mathsf{Recon}(\mathsf{crs}, \cdot)$ *is linear in its inputs.*
  - ***Succinct Shares:*** *There exists a polynomial* p *and a sublinear function* $g(x) \in o(x)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *with* ℓ_{inp}*-inputs and* ℓ_{out}*-outputs, the output length of* Enc *is at most* $p(m\lambda\ell_{\mathsf{inp}})g(|\mathsf{C}|)$ *and the output length of* Eval *is at most* $p(\lambda\ell_{\mathsf{out}})g(|\mathsf{C}|)$*.*
  - ***Succinct Verification:*** *There exists a polynomial* p *and a sublinear function* $g(x) \in o(x)$ *such that for every* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *with* ℓ_{inp}*-inputs and* ℓ_{out}*-outputs, the runtime of* LVerify *and* PVerify *are at most* $p(\lambda(\ell_{\mathsf{inp}} + \ell_{\mathsf{out}}))g(|\mathsf{C}|)$*. Note that the verification time is* independent *of the number of servers* m*.*
  - ***Circuit Independent Key Generation:*** KeyGen *is circuit independent i.e., the keys output by* KeyGen *can be used to evaluate any circuit in* $\mathbb{C}$*.*
- ***Correctness:*** *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all integer-valued polynomials* $\ell := \ell(\lambda)$*, circuits* $\mathsf{C} \in \mathbb{C}$*, inputs* $\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n \in R^{\ell}$*, and* $\lambda \in \mathbb{N}$*, we have*

$$\Pr\left[\begin{array}{c} \mathbf{y} \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n) \quad \vee \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \neq 1 \quad \vee \\ \exists i \in [1, n],\ \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_i, \mathbf{y}, \mathsf{st}_i, \pi_i) \neq 1 \end{array}\right] \leq \mathsf{negl}(\lambda),$$

  where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}, \mathsf{C})$*,* $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m)$ *is generated using* KeyGen*, each* $(\mathsf{st}_i, \widetilde{\mathbf{x}}_i) \leftarrow \mathsf{Enc}(\mathsf{crs}, i, \mathsf{pk}, \mathbf{x}_i)$*,* $\left[\mathsf{out}_{\mathsf{pub}}\right]$ *and each* $[\mathsf{out}_i]$ *are obtained by running* Eval *with each server's evaluations key,* $(\mathbf{y}, \pi_{\mathsf{pub}})$ *is obtained by reconstructing* $\left[\mathsf{out}_{\mathsf{pub}}\right]$ *and each* π_i *is obtained by reconstructing* $[\mathsf{out}_i]$*.*
- ***Privacy:*** *There exists a* PPT *simulator* Sim *such that for all polynomial size adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$*,*

$$\left|\Pr\left[\mathsf{Privacy}^{0}_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^{\lambda}) = 1\right] - \Pr\left[\mathsf{Privacy}^{1}_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^{\lambda}) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

  where $\mathsf{Privacy}^{b}_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^{\lambda})$ *is defined in Figure 7.*
- ***Local Soundness:*** *For every polynomial size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *we have*

$$\Pr\left[\begin{array}{c} \\ (\mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_i, \mathbf{y}) \notin \mathsf{SAT} \\ \wedge \\ \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_i, \mathbf{y}, \mathsf{st}_i, \pi_i) = 1 \\ \\ \end{array} \middle| \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}, \mathsf{C}) \\ (\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{KeyGen}(1^{\lambda}) \\ (\mathbf{x}_{\mathsf{pub}}, i, \mathbf{x}_i, \mathsf{st}_a) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \\ (\mathsf{st}_i, \widetilde{\mathbf{x}}_i) \leftarrow \mathsf{Enc}(\mathsf{crs}, i, \mathsf{pk}, \mathbf{x}_i) \\ (\left[\mathsf{out}_{\mathsf{pub}}\right], [\mathsf{out}_i]) \leftarrow \mathcal{A}(\mathsf{st}_a, \widetilde{\mathbf{x}}_i) \\ (\mathbf{y}, \pi_{\mathsf{pub}}) \leftarrow \mathsf{Recon}(\mathsf{crs}, \left[\mathsf{out}_{\mathsf{pub}}\right]) \\ \pi_i \leftarrow \mathsf{Recon}(\mathsf{crs}, [\mathsf{out}_i]) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Functionality $\mathcal{F}_{\mathsf{crs}}^{\mathsf{MveHSS}}$

$\mathcal{F}_{\mathsf{crs}}^{\mathsf{MveHSS}}$ is parameterized by a scheme MveHSS and proceeds as follows with n clients and m servers each with common input a circuit C, and an adversary Sim corrupting a subset $\mathcal{I} \subset [1, m]$ of servers and a subset $\mathcal{I}_{\mathsf{c}} \subseteq [1, n]$ of clients.

1: It receives λ and C from honest clients and servers and computes $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}, \mathsf{C})$.

2: It then sends crs to Sim and all clients and servers.

Figure 8: CRS functionality.

Functionality $\mathcal{F}_{\mathsf{pkgen}}^{\mathsf{MveHSS}}$

$\mathcal{F}_{\mathsf{pkgen}}^{\mathsf{MveHSS}}$ is parameterized by an MveHSS scheme MveHSS and proceeds as follows with n clients and m servers each with common input a circuit C, and an adversary Sim corrupting a subset $\mathcal{I} \subset [1, m]$ of servers and a subset $\mathcal{I}_{\mathsf{c}} \subseteq [1, n]$ of clients.

1: It receives λ from honest clients and servers and computes $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{KeyGen}(1^{\lambda})$.

2: It sends pk and $\{\mathsf{ek}_i\}_{i \in \mathcal{I}}$ to Sim and receives status from Sim. If status = abort, it sends $\perp$ to all servers else it sends pk to all clients and servers and ek_i to the i-th server.

Figure 9: Key generation functionality.

– ***Public Soundness:*** *For all polynomial size adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}$ we have*

$$\Pr\left[\begin{array}{c} (\mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}) \notin \mathsf{SAT} \\ \wedge \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}, \mathsf{C}) \\ (\mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Public and local knowledge soundness, as well as non-adaptive soundness and privacy, can also be defined for MveHSS, similar to how they're defined for ve-HSS in Section 5. We omit the explicit definitions here, since they can be easily obtained by adapting Definition 8.

Note that privacy in Definition 8 is defined with respect to semi-honest clients since the encryption of each input is computed by the game and not the adversary in Figure 7. We next define a stronger notion of MveHSS that considers an adversary that maliciously corrupts both clients and servers.

Functionality $\mathcal{F}_{\mathsf{mpc}}$

$\mathcal{F}_{\mathsf{mpc}}$ proceeds as follows with n clients each holding a private input $\mathbf{x}_i$ and common public inputs $(\mathsf{C}, \mathbf{x}_{\mathsf{pub}})$, and an adversary Sim corrupting a subset $\mathcal{I}_{\mathsf{c}} \subseteq [1, n]$ of the clients.

1: It receives the circuit C and the public input $\mathbf{x}_{\mathsf{pub}}$ from honest clients and sends $(\mathsf{C}, \mathbf{x}_{\mathsf{pub}})$ to Sim. It then receives the private circuit inputs from all clients. Let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ denote the tuple of private inputs.

2: It computes $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ and sends it to Sim and receives $\mathcal{S} \subseteq [1, n]$ from Sim. For every $i \in [1, n]$, it sends $\perp$ to the i-th client if $i \in \mathcal{S}$ else it sends $\mathbf{y}$ to it.

Figure 10: Functionality for ideal world definition of strong privacy in Definition 9.

Protocol $\Pi^{\mathsf{MveHSS}}_{\mathsf{mpc}}$

$\Pi^{\mathsf{MveHSS}}_{\mathsf{mpc}}$ is parameterized by a scheme MveHSS and is described in the $\left\{\mathcal{F}^{\mathsf{MveHSS}}_{\mathsf{crs}}, \mathcal{F}^{\mathsf{MveHSS}}_{\mathsf{pkgen}}\right\}$-hybrid model (Figures 8 and 9). It runs with clients $\mathcal{P}_{\mathsf{c}} = \left\{P^{\mathsf{c}}_1, \ldots, P^{\mathsf{c}}_n\right\}$ and servers $\mathcal{P} = \{P_1, \ldots, P_m\}$ where P^{c}_i has private input $\mathbf{x}_i$ and all clients and servers have the public input $\mathbf{x}_{\mathsf{pub}}$ and the circuit C. The protocol runs in the presence of a static adversary $\mathcal{A}$ that corrupts a subset $\mathcal{I} \subset [1, m]$ of servers and a subset $\mathcal{I}_{\mathsf{c}} \subseteq [1, n]$ of clients.

1: Each client and server invokes $\mathcal{F}^{\mathsf{MveHSS}}_{\mathsf{crs}}$ with circuit C to receive crs.
2: Each client and server invokes $\mathcal{F}^{\mathsf{MveHSS}}_{\mathsf{pkgen}}$ to receive pk. For each $i \in [1, m]$, the i-th server P_i additionally receives its evaluation key ek_i.
3: Each client P^{c}_i encrypts its input as $(\mathsf{st}_i, \widetilde{\mathbf{x}_i}) \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pk}, \mathbf{x}_i)$ and broadcasts $\widetilde{\mathbf{x}_i}$.
4: Each server P_i locally computes $\left(\left[\mathsf{out}_{\mathsf{pub}}\right]_i, [\mathsf{out}_1]_i, \ldots, [\mathsf{out}_n]_i\right) \coloneqq \mathsf{Eval}\left(\mathsf{crs}, \mathsf{ek}_i, \mathbf{x}_{\mathsf{pub}}, \widetilde{\mathbf{x}_1}, \ldots, \widetilde{\mathbf{x}_n}\right)$.
5: Each server P_i sends $\left[\mathsf{out}_{\mathsf{pub}}\right]_i$ to all clients and $\left[\mathsf{out}_j\right]_i$ to P^{c}_j, for each $j \in [1, n]$.
6: Each client P^{c}_i locally computes $(\mathbf{y}, \pi_{\mathsf{pub}}) \coloneqq \mathsf{Recon}\left(\mathsf{crs}, \left[\mathsf{out}_{\mathsf{pub}}\right]\right)$ and $\pi_i \coloneqq \mathsf{Recon}(\mathsf{crs}, [\mathsf{out}_i])$.
7: Each client P^{c}_i locally computes $\mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathsf{st}_i, \mathbf{y}, \pi_i) \rightarrow b_i$. It outputs $\mathbf{y}$ if $b_i = 1$ and outputs $\perp$ otherwise.

Figure 11: Protocol for real world definition of strong privacy in Definition 9.

Definition 9 (MveHSS With Strong Privacy)**.** *An m-server MveHSS scheme* MveHSS *for a class of circuits $\mathbb{C}$ over a ring R is said to have* strong privacy *if for every polynomial size adversary $\mathcal{A}$ there exists a simulator* Sim *such that*

$$\left\{\mathsf{Real}^{\mathcal{I},\mathcal{I}_{\mathsf{c}}}_{\mathcal{A}(z)}(1^\lambda, z, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)\right\} \overset{c}{\approx} \left\{\mathsf{Ideal}^{\mathcal{I},\mathcal{I}_{\mathsf{c}}}_{\mathsf{Sim}(z)}(1^\lambda, z, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)\right\}$$

where the ensembles are parameterized by $\lambda \in \mathbb{N}$, circuit $\mathsf{C} \in \mathbb{C}$, an integer-valued polynomial n, inputs $\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n \in R^*$, sets $\mathcal{I} \subset [1, m]$ and $\mathcal{I}_{\mathsf{c}} \subseteq [1, n]$, and auxiliary information $z \in \{0,1\}^*$. $\mathsf{Real}^{\mathcal{I},\mathcal{I}_{\mathsf{c}}}_{\mathcal{A}(z)}$ corresponds to the view of $\mathcal{A}$ and output of honest clients in $\Pi^{\mathsf{MveHSS}}_{\mathsf{mpc}}$ (Figure 11). $\mathsf{Ideal}^{\mathcal{I},\mathcal{I}_{\mathsf{c}}}_{\mathsf{Sim}(z)}$ corresponds to the output of the simulator and honest clients upon interacting with the $\mathcal{F}_{\mathsf{mpc}}$ functionality (Figure 10).

Since the real and the ideal world ensembles include the output of honest parties, strong privacy captures correctness as defined in Section 6.2.

Remark 8 (Publicly Auditable MPC)**.** *MveHSS with strong privacy implies a two-round client-server MPC with selective abort, assuming trusted setup. This follows immediately from the definition of strong privacy. Moreover, publishing $\mathbf{y}$ and the proof π_{pub} onto a bulletin board allows any external party to verify the correctness of $\mathbf{y}$ without providing any additional information about the private inputs of clients. Thus, strong privacy implies the stronger notion of MPC with public auditability [BDO14], with succinct public proofs.*

6.1 Construction With Strong Privacy

In this section, we present our construction for Multi-Client HSS with verifiable evaluation with strong privacy. In Section 6.2, we discuss how the construction can be simplified when security is required only against semi-honest clients.

Our MveHSS construction follows a similar approach as our ve-HSS construction where clients first use an HSS scheme to share their inputs followed by each server evaluating a circuit C and the non-linear

phase of a Splittable zkSNARG for C using the HSS scheme to obtain its share of the outputs. Servers then evaluate the linear phase of the zkSNARG over their shares to obtain shares of the proofs and then reconstruct the circuit output and proofs towards the clients. To support multiple clients, we use a public-key HSS scheme which allows any client holding a public key pk to encrypt its input for HSS evaluation. However, as discussed in Section 2, there are a number of changes we need to make to our approach to ensure security of the construction.

Semi-malicious correctness. Firstly, we require a stronger correctness property from public-key HSS schemes, called *semi-malicious correctness*, defined below. This requires that correctness of the scheme holds even when the randomness used for encrypting inputs is adversarially chosen. This is because, in our construction, the adversary computes encryptions of corrupt clients' inputs possibly based on the ciphertexts sent by honest clients. This stronger notion of adaptive correctness can be realized in the random oracle model for most HSS schemes, as discussed in [RS21, ADOS22].

Definition 10 (HSS Scheme with Semi-malicious Correctness)**.** *An m-server public-key HSS scheme* $\mathsf{HSS} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval})$ *for a class of circuits $\mathbb{C}_{\mathsf{hss}}$ over a ring R is said to have semi-malicious correctness if for all polynomial size adversaries, there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{hss}}$, we have*

$$\Pr\left[\begin{array}{c} [\mathsf{y}]_1 + \ldots + [\mathsf{y}]_m \\ \neq \\ \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, x_1, \ldots, x_n) \end{array} \;\middle|\; \begin{array}{r} (\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (\mathbf{x}_{\mathsf{pub}}, (x_1, r_1), \ldots, (x_n, r_n)) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \\ \widetilde{x_i} \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i; r_i),\ \forall i \in [1, n] \\ [\mathsf{y}]_i := \mathsf{Eval}(\mathsf{ek}_i, \mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \widetilde{x_1}, \ldots, \widetilde{x_n}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Randomness for zkSNARG. The randomness used to compute each zkSNARG proof must be uniformly random to ensure zero-knowledge, and consequently, the privacy of the inputs. This randomness is thus computed as the sum of the randomness shared by each client. This ensures uniform randomness as long as at least one client is honest; if all clients are corrupt, the adversary already knows all inputs, making the zero-knowledge requirement vacuous.

Well-formedness of ciphertexts. To ensure ciphertexts shared by clients are well-formed, we employ a NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ for the relation $\mathcal{R}_{\mathsf{hss}}$ (Figure 12), to verify that each ciphertext indeed corresponds to an encryption under pk. The NIZK statement consists of the HSS public key pk and a ciphertext $\widetilde{x}$ while the witness consists of the plaintext x and the randomness r used for encryption.

Correctness of output. Finally, to ensure correctness of the reconstructed output, we employ a BeDOZa-style MAC check [BDOZ11] as in [ADOS22]. Each party shares a MAC key that is used to authenticate their respective output. The client can then use the MAC to verify the output correctness. Let the HSS scheme support evaluating circuits over a ring R. To ensure soundness of the MAC check, we assume that the smallest prime divisor p of the ring order $|R|$ is such that $1/p$ is negligible in the security parameter.[19] If this is not the case, simply repeating the check with k independently sampled MAC keys, such that $1/p^k$ is negligible, suffices to ensure soundness with overwhelming probability.

Our MveHSS scheme is described in Figures 13 and 14 and its security is established in the following theorem. The theorem requires that for every circuit supported by the MveHSS scheme, the underlying HSS scheme supports two operations: (1) evaluating a MAC on the output under each client's key, and (2) evaluating the non-linear phase of the Splittable zkSNARG with randomness computed as the sum of inputs shared by clients.

[19] Formally, this requires that the HSS scheme be defined over a family of rings indexed by the security parameter, but we abuse notation for ease of exposition.

Functionality $\mathcal{F}_{\mathsf{NIZK}}$

$\mathcal{F}_{\mathsf{NIZK}}$ is parameterized by a public-key HSS scheme HSS. Let $\mathcal{R}_{\mathsf{hss}}$ denote the relation

$$\mathcal{R}_{\mathsf{hss}} = \{(\mathsf{pk}, \mathcal{M}, \widetilde{x}; x, r) \mid \widetilde{x} = \mathsf{HSS.Enc}(\mathsf{pk}, x; r) \wedge x \in \mathcal{M}\}$$

Proof: On input (prove, x, w) from a party P, check if $(x; w) \in \mathcal{R}_{\mathsf{hss}}$ and if so send (prove, P, x) to Sim. Upon receiving (proof, π) from Sim, store (x, w, π) and send (proof, π) to P.

Verify: On input $(\mathsf{verify}, x, \pi)$ from any party V, do the following.

- Check if (x, w, π) is stored for some w and if not send $(\mathsf{verify}, V, x, \pi)$ to Sim. Upon receiving $(\mathsf{witness}, w)$ from Sim store (x, w, π).
- Send $(\mathsf{verification}, x, \pi, b)$ to V where (x, w, π) is stored and $b = 1$ iff $(x; w) \in \mathcal{R}_{\mathsf{hss}}$.

Figure 12: NIZK Functionality

Theorem 15. *Let* HSS *be an m-server public-key HSS scheme with semi-malicious correctness for a class of circuits $\mathbb{C}_{\mathsf{hss}}$ over a finite commutative ring R and let* SNARG *be a $\mathbb{C}_{\mathsf{hss}}$-simple Splittable zkSNARG for $\mathbb{C}$-circuit satisfiability such that the following holds:*

- $\mathbb{C} \subseteq \mathbb{C}_{\mathsf{hss}}$,
- *For every ℓ-input circuit $\mathsf{C} \in \mathbb{C}_{\mathsf{hss}}$, for every integer-valued polynomial $n(\cdot)$, and every linear function $f : R^{n(\ell)} \to R^{\ell}$, the circuit $\hat{\mathsf{C}}$ computing $\hat{\mathsf{C}}(\mathbf{x}) = \mathsf{C}(f(\mathbf{x}))$ for $\mathbf{x} \in R^{n(\ell)}$ is in $\mathbb{C}_{\mathsf{hss}}$, and*
- *For every ℓ-input circuit $\mathsf{C} \in \mathbb{C}$ and for every integer-valued polynomial $n(\cdot)$, the circuit $\hat{\mathsf{C}}$ computing $\hat{\mathsf{C}}(\mathbf{x}, K_1, \ldots, K_{n(\ell)}) = (K_1 \cdot \mathsf{C}(\mathbf{x}), \ldots, K_{n(\ell)} \cdot \mathsf{C}(\mathbf{x}))$ for $\mathbf{x} \in R^{\ell}$ and $K_1, \ldots, K_{n(\ell)} \in R$ is in $\mathbb{C}_{\mathsf{hss}}$.*

Then, for any integer-valued polynomial $n := n(\lambda)$, there exists an n-client, m-server MveHSS scheme for $\mathbb{C}$ with strong privacy (Figures 13 and 14) in the $\mathcal{F}_{\mathsf{NIZK}}$-hybrid model that is black-box in HSS *and the linear phase of* SNARG.

Proof. We first show that the construction has strong privacy and then show that it satisfies the required soundness and efficiency properties. To ensure privacy, we assume that computation is performed over a ring R such that the smallest prime divisor p of its order $|R|$ is such that $1/p$ is negligible in the security parameter. If this is not the case, modifying the construction to repeat the check with k independently sampled MAC keys, such that $1/p^k$ is negligible, suffices to ensure soundness. Since $k \leq \lambda$ suffices for any ring R, this only results in a $\mathsf{poly}(\lambda)$ increase in each input and output share size.

Privacy. Intuitively, privacy of the honest clients' inputs follow from the security of HSS, the zero knowledge property of the proofs computed using $\mathcal{F}_{\mathsf{NIZK}}$, and the zero-knowlege property of the Splittable zkSNARG. Correctness of the scheme follows from the soundness of proofs computed using $\mathcal{F}_{\mathsf{NIZK}}$, correctness of HSS and the zkSNARG, and the soundness of the MAC check. An adversary does not gain any advantage by deviating from the protocol and learning if local verification was successful for honest clients due to the DP-robust verification property of the zkSNARG.

We next formally prove that the construction is secure against any polynomial size adversary $\mathcal{A}$ by describing a simulator Sim for $\mathcal{A}$ and arguing indistinguishability of $\mathcal{A}$'s view and the honest clients' outputs in the real and ideal worlds. Without loss of generality, we consider an adversary that corrupts all but one server and denote the index of the honest server by i^*.

Simulator:

Multi-Client HSS with Verifiable Evaluation

Public parameters. Let HSS be an m-server public-key HSS scheme that supports evaluating circuits over a finite commutative ring R and let SNARG be a Splittable zkSNARG. Let ℓ_{rand} denote the number of random elements from R required by the zkSNARG prover.

Notation. For a circuit C, let $\mathsf{C}_{\mathsf{pub}}$ and C_i denote the circuit satisfiability instances for the public proof and the i-th client's proof, respectively i.e., when $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$ then $(\mathsf{C}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}) \in \mathsf{SAT}$ and $(\mathsf{C}_i, \mathbf{x}_{\mathsf{pub}} \| \mathbf{x}_i, \mathbf{y}) \in \mathsf{SAT}$. Let $\mathsf{C}^{\mathsf{pub}}_{\mathsf{NL}}$ and $\mathsf{C}^{i}_{\mathsf{NL}}$ denote the circuits that compute the non-linear phase $\mathsf{Prove}_{\mathsf{NL}}$ of the Splittable zkSNARG for $\mathsf{C}_{\mathsf{pub}}$ and C_i, respectively. Let $\hat{\mathsf{C}}_{\mathsf{NL}}$ denote the circuit that evaluates C_{NL} using randomness derived from the sum of the last n inputs. For a circuit C such that $\mathbf{y} = \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$, let $\widehat{\mathsf{C}}$ denote the augmented circuit such that $(\mathbf{y}, \Gamma^1_{\mathbf{y}}, \ldots, \Gamma^n_{\mathbf{y}}) = \widehat{\mathsf{C}}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n, K_1, \ldots, K_n)$, where $\Gamma^i_{\mathbf{y}} = \mathbf{y} \cdot K_i$ for each $i \in [1, n]$.

$\mathsf{MveHSS.Setup}(1^\lambda, \mathsf{C})$

$\mathsf{crs}_{\mathsf{pub}} \leftarrow \mathsf{SNARG.Setup}(1^\lambda, \mathsf{C}_{\mathsf{pub}})$
for $i \in [1, n]$
  $\mathsf{crs}_i \leftarrow \mathsf{SNARG.Setup}(1^\lambda, \mathsf{C}_i)$
$\mathsf{crs} := (\mathsf{crs}_{\mathsf{pub}}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{C})$
return crs

$\mathsf{MveHSS.KeyGen}(1^\lambda, \mathsf{C})$

$(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{HSS.KeyGen}(1^\lambda)$
return $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m)$

$\mathsf{MveHSS.Enc}(\mathsf{crs}, i, \mathsf{pk}, \mathbf{x})$

$K \leftarrow R$
$\mathbf{r}_{\mathsf{pub}}, \mathbf{r}_1, \ldots, \mathbf{r}_n \leftarrow R^{\ell_{\mathsf{rand}}}$
foreach $y \in \{\mathbf{x}, K, \mathbf{r}_{\mathsf{pub}}, \mathbf{r}_1 \ldots, \mathbf{r}_n\}$
  $\rho \leftarrow \{0,1\}^\lambda$
  $\mathsf{ct}_y \leftarrow \mathsf{HSS.Enc}(\mathsf{pk}, y; \rho)$
  Send $(\mathsf{prove}, (\mathsf{pk}, R, \mathsf{ct}_y), (y, \rho))$ to $\mathcal{F}_{\mathsf{NIZK}}$ and receive $(\mathsf{proof}, \pi^{\mathsf{inp}}_{\mathbf{y}})$
$\widetilde{\mathbf{x}} := ((\mathsf{ct}_{\mathbf{x}}, \pi^{\mathsf{inp}}_{\mathbf{x}}), (\mathsf{ct}_K, \pi^{\mathsf{inp}}_K), (\mathsf{ct}_{\mathbf{r}_{\mathsf{pub}}}, \pi^{\mathsf{inp}}_{\mathbf{r}_{\mathsf{pub}}}), (\mathsf{ct}_{\mathbf{r}_1}, \pi^{\mathsf{inp}}_{\mathbf{r}_1}), \ldots, (\mathsf{ct}_{\mathbf{r}_n}, \pi^{\mathsf{inp}}_{\mathbf{r}_n}))$
$\mathsf{st} := K$
return $(\widetilde{\mathbf{x}}, \mathsf{st})$

$\mathsf{MveHSS.Eval}(\mathsf{crs}, \mathsf{ek}_i, \mathbf{x}_{\mathsf{pub}}, \widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_n)$

parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{pub}}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{C})$
parse each $\widetilde{\mathbf{x}}_i = ((\mathsf{ct}_{\mathbf{x}_i}, \pi^{\mathsf{inp}}_{\mathbf{x}_i}), (\mathsf{ct}_{K_i}, \pi^{\mathsf{inp}}_{K_i}), (\mathsf{ct}_{\mathbf{r}_{\mathsf{pub},i}}, \pi^{\mathsf{inp}}_{\mathbf{r}_{\mathsf{pub},i}}), (\mathsf{ct}_{\mathbf{r}_{1,i}}, \pi^{\mathsf{inp}}_{\mathbf{r}_{1,i}}), \ldots, (\mathsf{ct}_{\mathbf{r}_{n,i}}, \pi^{\mathsf{inp}}_{\mathbf{r}_{n,i}}))$
Verify each$(\mathsf{ct}_y, \pi^{\mathsf{inp}}_y)$ using $\mathcal{F}_{\mathsf{NIZK}}$ and abort if verification fails
$\left([\mathbf{y}]_i, \left[\Gamma^1_{\mathbf{y}}\right]_i, \ldots, \left[\Gamma^n_{\mathbf{y}}\right]_i\right) :=$ $\mathsf{HSS.Eval}(\mathsf{ek}_i, \widehat{\mathsf{C}}, \mathbf{x}_{\mathsf{pub}}, \mathsf{ct}_{\mathbf{x}_1}, \ldots, \mathsf{ct}_{\mathbf{x}_n}, \mathsf{ct}_{K_1}, \ldots, \mathsf{ct}_{K_n})$
$\left[M_{\mathsf{pub}}\right]_i := \mathsf{HSS.Eval}(\mathsf{ek}_i, \hat{\mathsf{C}}^{\mathsf{pub}}_{\mathsf{NL}}, \mathbf{x}_{\mathsf{pub}}, \mathsf{ct}_{\mathbf{x}_1}, \ldots, \mathsf{ct}_{\mathbf{x}_n}, \mathsf{ct}_{\mathbf{r}_{\mathsf{pub},1}}, \ldots, \mathsf{ct}_{\mathbf{r}_{\mathsf{pub},n}})$
$\left[\pi_{\mathsf{pub}}\right]_i := \mathsf{SNARG.Prove}_{\mathsf{L}}\left(\mathsf{crs}_{\mathsf{pub}}, \left[M_{\mathsf{pub}}\right]_i\right)$
$\left[\mathsf{out}_{\mathsf{pub}}\right]_i := \left([\mathbf{y}]_i, \left[\pi_{\mathsf{pub}}\right]_i\right)$
for $j \in [1, n]$
  $\left[M_j\right]_i := \mathsf{HSS.Eval}(\mathsf{ek}_i, \hat{\mathsf{C}}^{j}_{\mathsf{NL}}, \mathbf{x}_{\mathsf{pub}}, \mathsf{ct}_{\mathbf{x}_1}, \ldots, \mathsf{ct}_{\mathbf{x}_n}, \mathsf{ct}_{\mathbf{r}_{j,1}}, \ldots, \mathsf{ct}_{\mathbf{r}_{j,n}})$
  $\left[\pi^{\mathsf{loc}}_j\right]_i := \mathsf{SNARG.Prove}_{\mathsf{L}}\left(\mathsf{crs}_j, \left[M_j\right]_i\right)$
  $\left[\mathsf{out}_j\right]_i := \left(\left[\Gamma^j_{\mathbf{y}}\right]_i, \left[\pi^{\mathsf{loc}}_j\right]_i\right)$
return $\left(\left[\mathsf{out}_{\mathsf{pub}}\right]_i, [\mathsf{out}_1]_i, \ldots, [\mathsf{out}_n]_i\right)$

Figure 13: Multi-client HSS with verifiable evaluation and strong privacy from HSS and Splittable zkSNARG. The reconstruction and verification algorithms are described in Figure 14.

Multi-Client HSS with Verifiable Evaluation (continued)

$\mathsf{MveHSS.Recon}(\mathsf{crs}, [\mathsf{out}]_1, \ldots, [\mathsf{out}]_m)$

1 : **parse** each $[\mathsf{out}]_i = ([\mathbf{z}]_i, [\pi]_i)$

2 : $\mathbf{z} := \sum_{i=1}^{m} [\mathbf{z}]_i$

3 : $\pi := \sum_{i=1}^{m} [\pi]_i$

4 : **return** $(\mathbf{z}, \pi)$

$\mathsf{MveHSS.LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_i, \mathbf{y}, \mathsf{st}_i, \pi_i)$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{pub}}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$

2 : **parse** $\mathsf{st}_i = K_i$

3 : **parse** $\pi_i = (\Gamma_{\mathbf{y}}^i, \pi_i^{\mathsf{loc}})$

4 : **if** $K_i \cdot \mathbf{y} \neq \Gamma_{\mathbf{y}}^i \vee$
$\mathsf{SNARG.Verify}(\mathsf{crs}_i, \mathbf{x}_{\mathsf{pub}} \| \mathbf{x}_i, \mathbf{y}, \pi_i^{\mathsf{loc}}) \neq 1$

5 : **return** 0

6 : **else return** 1

$\mathsf{MveHSS.PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}})$

1 : **parse** $\mathsf{crs} = (\mathsf{crs}_{\mathsf{pub}}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$

2 : **return** $\mathsf{SNARG.Verify}(\mathsf{crs}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}})$

Figure 14: Reconstruction and verification algorithms for multi-client HSS with verifiable evaluation and strong privacy. The remaining algorithms are described in Figure 13.

1. It receives the circuit C and the public input $\mathbf{x}_{\mathsf{pub}}$ from $\mathcal{F}_{\mathsf{mpc}}$. If all clients are corrupt, it computes $\mathsf{crs} \leftarrow \mathsf{MveHSS.Setup}(1^\lambda, \mathsf{C})$. Else, it simulates the CRS for the public proof $\mathsf{crs}_{\mathsf{pub}}$ and the CRS for the local proofs of corrupt clients $\{\mathsf{crs}_i\}_{i \in \mathcal{I}_c}$ using the zkSNARG simulator, and computes the CRS for the local proofs of honest clients $\{\mathsf{crs}_i\}_{i \notin \mathcal{I}_c}$ honestly, using SNARG.Setup. It sends $\mathsf{crs} = (\mathsf{crs}_{\mathsf{pub}}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{crs}}$.

2. It computes $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_m) \leftarrow \mathsf{HSS.KeyGen}(1^\lambda)$ and sends the public-key pk and the evaluation keys of corrupt servers $\{\mathsf{ek}_i\}_{i \in \mathcal{I}}$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{pkgen}}$. It receives status from $\mathcal{A}$. If status = abort then it sends $\mathcal{S} = [1, n] \setminus \mathcal{I}_c$ to $\mathcal{F}_{\mathsf{mpc}}$ and halts.

3. It simulates the inputs of honest clients for each $i \in [1, n] \setminus \mathcal{I}_c$ by computing each HSS ciphertext in $\widetilde{\mathbf{x}}_i$ as an encryption of 0. It sends the corresponding statements under $\mathcal{R}_{\mathsf{hss}}$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{NIZK}}$ and receives the NIZK proof of well-formedness for each ciphertext. It returns 1 on behalf of $\mathcal{F}_{\mathsf{NIZK}}$ on subsequent verification queries on these proofs. It then sends $\{\widetilde{\mathbf{x}}_i\}_{i \notin \mathcal{I}_c}$, consisting of the encryptions of 0 and the NIZK proofs, to $\mathcal{A}$.

4. For each $i \in \mathcal{I}_c$, it obtains $\{\widetilde{\mathbf{x}}_i\}_{i \in \mathcal{I}_c}$ broadcast by $\mathcal{A}$. On behalf of each honest server, it emulates a verification query to $\mathcal{F}_{\mathsf{NIZK}}$ (where the simulator runs the steps of $\mathcal{F}_{\mathsf{NIZK}}$ honestly) for every NIZK proof sent by corrupt clients. If there exists a proof that does not verify, it sends $\mathcal{S} = [1, n] \setminus \mathcal{I}_c$ to $\mathcal{F}_{\mathsf{mpc}}$ and halts. At this point, the simulator obtains the witness $(\mathbf{x}_i, K_i, \mathbf{r}_{\mathsf{pub},i}, \mathbf{r}_{1,i}, \ldots, \mathbf{r}_{n,i})$ for each corrupt client $i \in \mathcal{I}_c$, either by receiving it from corrupt clients during the proof phase or from $\mathcal{A}$ during the verification phase.

5. It computes the output share of the i-th corrupt server, for each $i \in \mathcal{I}$, as

$$\left([\mathbf{y}]_i, [\pi_{\mathsf{pub}}]_i, \{[\pi_j]_i\}_{j=1}^n, \left\{\left[\Gamma_{\mathbf{y}}^j\right]_i\right\}_{j=1}^n\right) \leftarrow \mathsf{Eval}(\mathsf{crs}, \mathsf{ek}_i, \mathbf{x}_{\mathsf{pub}}, \widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_n).$$

6. It sends $\{\mathbf{x}_i\}_{i\in\mathcal{I}_c}$ to $\mathcal{F}_{\mathsf{mpc}}$ and obtains the output $\mathbf{y}$. For every corrupt client $i \in \mathcal{I}_c$, it computes $\Gamma^i_{\mathbf{y}} = \mathbf{y}\cdot K_i$. If all clients are corrupt, it computes

$$M_{\mathsf{pub}} \coloneqq \mathsf{SNARG.Prove}_{\mathsf{NL}}(\mathsf{crs}_{\mathsf{pub}}, \mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, (\mathbf{x}_1, \ldots, \mathbf{x}_n); \mathbf{r}_{\mathsf{pub},1} + \ldots + \mathbf{r}_{\mathsf{pub},n})$$
$$\pi_{\mathsf{pub}} \coloneqq \mathsf{SNARG.Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{pub}}).$$

Similarly, it computes $\{\pi^{\mathsf{loc}}_i\}^n_{i=1}$. Else, if at least one client is honest, it simulates π_{pub} and $\{\pi^{\mathsf{loc}}_i\}_{i\in\mathcal{I}}$ using the zkSNARG simulator.

7. It reverse samples the output share of the honest server $[\mathbf{y}]_{i^*}$, $[\pi_{\mathsf{pub}}]_{i^*}$, $\left[\Gamma^i_{\mathbf{y}}\right]_{i^*}$, and $[\pi^{\mathsf{loc}}_i]_{i^*}$. It then sends the shares of honest servers in $[\mathsf{out}_{\mathsf{pub}}]$ and $\{[\mathsf{out}_i]\}_{i\in\mathcal{I}_c}$ to $\mathcal{A}$ and receives $\left[\overline{\mathsf{out}_i}\right]_{\mathcal{I}} = \left(\left[\overline{\Gamma^i_y}\right]_{\mathcal{I}_c}, \left[\overline{\pi^{\mathsf{loc}}_i}\right]_{\mathcal{I}}\right)$ and $\left[\overline{\mathsf{out}^i_{\mathsf{pub}}}\right]_{\mathcal{I}} = \left([\overline{\mathbf{y}_i}]_{\mathcal{I}}, \left[\overline{\pi^i_{\mathsf{pub}}}\right]_{\mathcal{I}}\right)$ for each $i \notin \mathcal{I}_c$ from $\mathcal{A}$ ($\mathsf{out}_{\mathsf{pub}}$ is indexed by i because the output shares sent to each client might be different).

8. For every honest client $i \in [1,n] \setminus \mathcal{I}_c$ it does the following.

   (a) It computes $[\epsilon_i]_{\mathcal{I}} = [\overline{\mathbf{y}_i}]_{\mathcal{I}} - [\mathbf{y}]_{\mathcal{I}}$ and sets $[\epsilon_i]_j = 0$ for every $j \notin \mathcal{I}$. It then computes $\epsilon_i \coloneqq \sum_{j=1}^m [\epsilon_i]_j$.

   (b) It computes $[\delta_i]_{\mathcal{I}} = \left[\overline{\Gamma^i_{\mathbf{y}}}\right]_{\mathcal{I}} - \left[\Gamma^i_{\mathbf{y}}\right]_{\mathcal{I}}$ and sets $[\delta_i]_j = 0$ for every $j \notin \mathcal{I}$. It then computes $\delta_i \coloneqq \sum_{j=1}^m [\delta_i]_j$.

   (c) It computes $[\eta_i]_{\mathcal{I}} = \left[\overline{\pi^{\mathsf{loc}}_i}\right]_{\mathcal{I}} - \left[\pi^{\mathsf{loc}}_i\right]_{\mathcal{I}}$ and sets $[\eta_i]_j = 0$ for every $j \notin \mathcal{I}$. It then computes $\eta_i \coloneqq \sum_{j=1}^m [\eta_i]_j$.

   It then computes $\mathcal{S}$ as the set of all indices $i \in [1,n] \setminus \mathcal{I}_c$ such that at least one among ϵ_i, δ_i or η_i is non-zero. It sends $\mathcal{S}$ to $\mathcal{F}_{\mathsf{mpc}}$ and outputs the value returned by $\mathcal{A}$.

We show using a hybrid argument that for any $\mathsf{C} \in \mathbb{C}$, $\mathsf{Real}^{\mathcal{I},\mathcal{I}_c}_{\mathcal{A}(z)}$ is indistinguishable from $\mathsf{Ideal}^{\mathcal{I},\mathcal{I}_c}_{\mathsf{Sim}(z)}$.

- Hyb_0: This corresponds to $\mathsf{Real}_{\mathcal{A}}(1^\lambda, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$ i.e., the simulator follows the steps of $\mathcal{F}_{\mathsf{mpc}}$ using the honest parties' inputs.

- Hyb_1: In this hybrid, the simulator computes output shares of honest servers as follows.

  1. On behalf of $\mathcal{F}_{\mathsf{NIZK}}$, it learns the input shared by corrupt clients, as described in step 4 above. It then computes $\mathbf{y}$, $\left\{\Gamma^i_{\mathbf{y}}\right\}^n_{i=1}$, π_{pub} and $\{\pi^{\mathsf{loc}}_i\}^n_{i=1}$ in the clear by evaluating the corresponding circuits on client inputs.
  2. It recomputes the output shares of corrupt servers as described in step 5 above.
  3. Finally, it reverse samples the shares of the honest server in $[\mathbf{y}]$, $\left\{\left[\Gamma^i_{\mathbf{y}}\right]\right\}^n_{i=1}$, $[\pi_{\mathsf{pub}}]$, and $\left\{\left[\pi^{\mathsf{loc}}_i\right]\right\}^n_{i=1}$ using the shares of the corrupt servers and the corresponding secrets.

  In Hyb_0, the honest servers' shares of the output were computed by evaluating their HSS shares of the inputs while in this hybrid they are reverse sampled from the actual outputs and the corrupt servers' shares of the output. Observe that if the ciphertexts sent by corrupt clients are not well-formed then the previous hybrid is identical to this hybrid since the honest servers abort in both cases. Conversely, the honest servers do not abort only when all NIZK proofs successfully verify, in which case it follows from the definition of $\mathcal{F}_{\mathsf{NIZK}}$ that all input ciphertexts are well-formed. From the properties of $\mathbb{C}_{\mathsf{hss}}$ and

$\mathbb{C}$ stated in the theorem statement, it follows that for each $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}, \widehat{\mathsf{C}}, \hat{\mathsf{C}}^{\mathsf{pub}}_{\mathsf{NL}}, \hat{\mathsf{C}}^{1}_{\mathsf{NL}}, \ldots, \hat{\mathsf{C}}^{n}_{\mathsf{NL}} \in \mathbb{C}_{\mathsf{hss}}$. This combined with the semi-malicious correctness of HSS and the linearity of additive secret sharing implies that $\mathsf{Hyb}_0 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1$.

- Hyb_2: In this hybrid, the shares of honest servers in $[\mathbf{y}]$ are computed using the output $\mathbf{y}$ returned by $\mathcal{F}_{\mathsf{mpc}}$ and the output shares of corrupt servers computed by the simulator.

  Observe that honest clients and the simulator send the same inputs to $\mathcal{F}_{\mathsf{mpc}}$ as the ones used to compute $\mathbf{y}$ in Hyb_1. It thus follows that $\mathsf{Hyb}_1 \overset{\mathsf{p}}{\equiv} \mathsf{Hyb}_2$.

- Hyb_3: In this hybrid, the simulator simulates HSS inputs of honest clients as described in step 3 above.

  Observe that the verification of the honest clients' inputs using $\mathcal{F}_{\mathsf{NIZK}}$ is always successful in both hybrids. Thus, the main difference between Hyb_2 and Hyb_3 is that in Hyb_2 the simulator used the honest clients' inputs to compute the ciphertexts sent to the adversary while in this hybrid the ciphertexts correspond to encryptions of 0. The security of HSS then implies that $\mathsf{Hyb}_2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3$ from a standard hybrid argument.

- Hyb_4: In this hybrid, the simulator computes $\mathsf{crs}_{\mathsf{pub}}, \pi_{\mathsf{pub}}, \left\{\mathsf{crs}_i, \pi^{\mathsf{loc}}_i\right\}_{i \in \mathcal{I}_{\mathsf{c}}}$ as described in steps 1 and 6 above. It then uses these proofs to compute the output shares of honest servers as described in step 7.

  If all clients are corrupt then Hyb_3 is identical to Hyb_4. The only difference compared to to the previous hybrid is that when at least one client is honest, the public proof π_{pub} and the local proofs of corrupt clients $\left\{\pi^{\mathsf{loc}}_i\right\}_{i \in \mathcal{I}_{\mathsf{c}}}$ are simulated using the zkSNARG simulator. Observe that the randomness used to compute these proofs corresponds to the sum of random inputs shared by each client. Thus, when at least one of the clients is honest, the randomness used to compute the proofs is uniformly distributed. Since $\mathcal{A}$'s view in Hyb_3 is independent of the random inputs shared by honest clients, it can be shown using a standard hybrid argument that $\mathsf{Hyb}_3 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_4$ from the zero-knowledge property of the zkSNARG.

- Hyb_5: In this hybrid, the output of all honest clients P^{c}_i such that $\epsilon_i \neq 0$ or $\delta_i \neq 0$ is set to $\perp$ where ϵ_i and δ_i are as computed by the simulator in step 8.

  Observe that

$$\overline{\mathbf{y}_i} = \mathbf{y} + \sum_{j=1}^{n} [\overline{\mathbf{y}_i}]_j - [\mathbf{y}]_j = \mathbf{y} + \epsilon_i.$$

  Similarly, we have $\overline{\Gamma^i_{\mathbf{y}}} = \Gamma^i_{\mathbf{y}} + \delta_i$. The difference between Hyb_4 and Hyb_5 is that in Hyb_4 the output of honest clients was $\perp$ if $\overline{\mathbf{y}_i} \cdot K_i \neq \overline{\Gamma^i_{\mathbf{y}}}$ while in Hyb_5 it is set based on ϵ_i and δ_i. We show that the MAC check fails with overhwelming probability in Hyb_4 when $\epsilon_i \neq 0$ or $\delta_i \neq 0$.

  Observe that if $\overline{\mathbf{y}_i} \cdot K_i = \overline{\Gamma^i_{\mathbf{y}}}$ then

$$\begin{aligned} & \overline{\mathbf{y}_i} \cdot K_i = \overline{\Gamma^i_{\mathbf{y}}} \\ \implies \quad & (\mathbf{y} + \epsilon_i) \cdot K_i = \Gamma^i_{\mathbf{y}} + \delta_i \\ \implies \quad & \epsilon_i \cdot K_i = \delta_i \end{aligned}$$

  where the second implication follows from the fact that $\Gamma^i_{\mathbf{y}} = \mathbf{y} \cdot K_i$ by definition of $\widehat{\mathsf{C}}$. Note that K_i is uniformly random over R for honest clients and $\mathcal{A}$'s view is independent of K_i. Thus, when $\epsilon_i \neq 0$ or $\delta_i \neq 0$, it follows that $\epsilon_i \cdot K_i = \delta_i$ with probability at most $1/p$, where p is the smallest prime divisor of $|R|$. This implies that $\overline{\mathbf{y}_i} \cdot K_i = \overline{\Gamma^i_{\mathbf{y}}}$ with probability at most $1/p$, which is negligible. Thus, $\mathsf{Hyb}_4 \overset{\mathsf{s}}{\approx} \mathsf{Hyb}_5$.

- Hyb_6: In this hybrid, the output of all honest clients P^{c}_i such that $\eta_i \neq 0$ is set to $\perp$ where η_i is as computed by the simulator in step 8.

As in the previous case, we have $\overline{\pi_i^{\mathsf{loc}}} = \pi_i^{\mathsf{loc}} + \eta_i$ for each $i \notin \mathcal{I}_{\mathsf{c}}$. If the output of any honest client P_i^{c} is $\perp$ in this hybrid but not $\perp$ in the previous hybrid then $\overline{\mathbf{y}_i}$ reconstructed by P_i^{c} is equal to $\mathbf{y}$ but the reconstructed local proof $\overline{\pi_i^{\mathsf{loc}}}$ is not equal to π_i^{loc}. Observe that the randomness used to compute the local proof of each honest client is uniformly distributed, since this randomness corresponds to the sum of random inputs shared by all clients. Since the adversary's view is independent of the random inputs shared by honest clients, the DP-robust verification property of SNARG (Lemma 7) implies that the output of P_i^{c} in Hyb_5 is $\perp$ with overwhelming probability when $\eta_i \neq 0$. It can then be shown using a standard hybrid argument that $\mathsf{Hyb}_5 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_6$.

– Hyb_7: In this hybrid, the outputs of honest clients is as received from $\mathcal{F}_{\mathsf{mpc}}$.

It is easy to see that if $\mathcal{F}_{\mathsf{pkgen}}$ sends $\perp$ or verification of corrupt clients' inputs using $\mathcal{F}_{\mathsf{NIZK}}$ fails then honest clients abort in Hyb_6 and Hyb_7. Moreover, if one among ϵ_i, δ_i or η_i is non-zero for a client P_i^{c} then $\mathcal{F}_{\mathsf{mpc}}$ sends $\perp$ to P_i^{c} since the simulator adds i to $\mathcal{S}$. Note that in this case, the output of honest clients is identical to that in the previous hybrid. Conversely, when $\epsilon_i = 0$, $\delta_i = 0$ and $\eta_i = 0$, the correctness of the MAC check and the DP-completeness property (Lemma 6) imply that $\mathsf{Hyb}_6 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_7$.

Hyb_7 corresponds to $\mathsf{Ideal}^{\mathcal{I},\mathcal{I}_{\mathsf{c}}}_{\mathsf{Sim}(z)}$ which in turn implies that

$$\left\{\mathsf{Real}^{\mathcal{I},\mathcal{I}_{\mathsf{c}}}_{\mathcal{A}(z)}(1^\lambda, z, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)\right\} \overset{\mathsf{c}}{\approx} \left\{\mathsf{Ideal}^{\mathcal{I},\mathcal{I}_{\mathsf{c}}}_{\mathsf{Sim}(z)}(1^\lambda, z, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n)\right\}$$

where the ensembles are parameterized by $\lambda \in \mathbb{N}$, circuit $\mathsf{C} \in \mathbb{C}$, an integer-valued polynomial n, inputs $\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_1, \ldots, \mathbf{x}_n \in R^*$, sets $\mathcal{I} \subset [1, m]$ and $\mathcal{I}_{\mathsf{c}} \subseteq [1, n]$, and auxiliary information $z \in \{0,1\}^*$.

Local soundness and public soundness of the construction follow immediately from the soundness of the zkSNARG. The construction has succinct share size and succinct verification since HSS is compact and SNARG has succinct proof size and verification. It then follows that the construction described in Figures 13 and 14 is an n-client, m-server MveHSS scheme for $\mathbb{C}$ with strong privacy. ■

6.2 Simpler Construction Without Strong Privacy

Our MveHSS construction can be simplified when strong privacy is not required. Specifically, we no longer need the $\mathcal{F}_{\mathsf{NIZK}}$ functionality and semi-malicious correctness of the public-key HSS scheme. In more detail, $\mathcal{F}_{\mathsf{NIZK}}$ serves two purposes in the previous construction: firstly, to ensure well-formedness of the ciphertexts sent by corrupt clients and secondly, it allows the simulator to extract the inputs of corrupt clients. However, if clients are only semi-honestly corrupt, then the well-formedness of ciphertexts is guaranteed since all clients follow the protocol and the simulator no longer needs to extract inputs. Thus, in this case, the construction no longer requires $\mathcal{F}_{\mathsf{NIZK}}$, and the Enc algorithm only requires clients to encrypt their inputs, MAC keys and their contribution to the randomness used for computing the Splittable zkSNARG. Moreover, when clients are semi-honest the randomness used for this encryption is uniformly distributed which in turn implies that the standard correctness of the public-key HSS scheme (as defined in Definition 2) suffices, as opposed to the stronger semi-malicious correctness.

Theorem 16. *Let* HSS *be an m-server public-key HSS scheme for a class of circuits* $\mathbb{C}_{\mathsf{hss}}$ *over a finite commutative ring R and let* SNARG *be a* $\mathbb{C}_{\mathsf{hss}}$*-simple Splittable zkSNARG for* $\mathbb{C}$*-circuit satisfiability such that* $\mathbb{C}_{\mathsf{hss}}$ *and* $\mathbb{C}$ *satisfy the same requirements as in Theorem 15. Then, for any integer-valued polynomial* $n := n(\lambda)$*, there exists an n-client, m-server MveHSS scheme for* $\mathbb{C}$ *that is black-box in* HSS *and the linear phase of* SNARG*.*

6.3 Instantiations

We now discuss instantiating the MveHSS template described in Figures 13 and 14 using the HSS schemes from prior works and the Splittable zkSNARGs discussed in Section 4.

Let $\mathsf{VP}[p]$ denote the set of polynomial size circuits over $\mathbb{Z}_p$ and let $\mathsf{VNC}^1[p]$ denote the set of polynomial size circuits over $\mathbb{Z}_p$ that have logarithmic depth in the number of inputs. As discussed in Section 5.1, we consider two classes of public-key HSS schemes. The first group consists of 2-server public-key HSS schemes from group based assumptions [BKS19, OSY21, RS21, ADOS22] that can be used to evaluate circuits in $\mathsf{VNC}^1[p]$, where $p \in O(2^\lambda)$. The second group consists of m-server public-key HSS schemes [BGI15, COS+22], where m is polynomial in the security parameter, that can be used to evaluate circuits in $\mathsf{VP}[p]$, for $p \in O(2^\lambda)$. Note that both groups satisfy the preconditions for Theorem 15 since $\mathsf{VNC}^1[p]$ and $\mathsf{VP}[p]$ are closed under linear transformations on the input and computing MACs on the output.

Groth16 based MveHSS. As discussed in Section 5.1, Theorem 9 implies that Groth16 is a $\mathsf{VNC}^1[p]$-simple SplittablezkSNARG for NC^1-circuit satisfiability and that it is a $\mathsf{VP}[p]$-simple Splittable zkSNARG for $\mathsf{P/poly}$ circuit satisfiability. This gives the following corollary of Theorem 15.

Corollary 17. *There exist the following MveHSS schemes with local and public knowledge soundness in the generic bilinear group model:*

- *A 2-server MveHSS scheme for* NC^1 *assuming the hardness of the LWE assumption,*
- *A 2-server MveHSS scheme for* NC^1 *assuming the hardness of the DCR assumption,*
- *A 2-server MveHSS scheme for* NC^1 *assuming the hardness of the DDH assumption and the small exponent assumption over class groups,*
- *An m-server MveHSS scheme for* $\mathsf{P/poly}$*, for any integer-valued polynomial $m(\cdot)$, assuming the existence of circular secure AFS-spooky encryption.*

Waters-Wu zkBARG based MveHSS. The following corollary of Theorem 15 follows from Theorem 11, which implies that the Waters-Wu zkBARG has non-adaptive soundness and is a $\mathsf{VNC}^1[N]$-simple zkSNARG for SIMD NC^1-circuit satisfiability and is a $\mathsf{VP}[N]$-simple zkSNARG for SIMD $\mathsf{P/poly}$-circuit satisfiability, where N is the order of the bilinear groups used for the zkBARG. Interestingly, unlike in the case of single client ve-HSS where non-adaptive soundness implied non-adaptive privacy, the privacy of the multi-client scheme is not affected by the soundness of the zkBARG. This is because the single client ve-HSS construction relied on the soundness of the zkBARG for ensuring correctness of the output while the multi-client construction uses MACs.

Corollary 18. *There exist the following MveHSS schemes with non-adaptive local and public soundness all of which assume the hardness of the subgroup decision assumption over composite-order bilinear groups:*

- *A 2-server MveHSS scheme for SIMD* NC^1 *additionally assuming the hardness of the LWE assumption,*
- *A 2-server MveHSS scheme for SIMD* NC^1 *additionally assuming the hardness of the DCR assumption,*
- *A 2-server MveHSS scheme for SIMD* NC^1 *additionally assuming the hardness of the DDH assumption and the small exponent assumption over class groups,*
- *An m-server MveHSS scheme for SIMD* $\mathsf{P/poly}$*, for any integer-valued polynomial $m(\cdot)$, additionally assuming the existence of circular secure AFS-spooky encryption.*

MveHSS with strong privacy. Instantiating an MveHSS scheme with strong privacy additionally requires a public-key HSS scheme with semi-malicious correctness and a primitive that realizes $\mathcal{F}_{\mathsf{NIZK}}$. As discussed in [RS21,ADOS22], semi-malicious correctness can be achieved for most 2-server public key HSS schemes in the random oracle model. On the other hand, to realize $\mathcal{F}_{\mathsf{NIZK}}$ while making only black-box use of cryptography, we use the NIZK scheme proposed by Abram et al. [ADOS22]. Specifically, they describe a 2-server public-key HSS scheme with semi-malicious correctness and a NIZK scheme that realizes $\mathcal{F}_{\mathsf{NIZK}}$ in the random oracle model assuming the hardness of the Decisional Diffie-Hellman (DDH) assumption, Decisional Cross-group Diffie-Hellman (DXDH) assumption, weak hidden order assumption and small exponent assumption over class groups. The HSS and NIZK schemes are black-box in the group. Furthermore, [ADOS22] also describe a one-round protocol realizing $\mathcal{F}_{\mathsf{pkgen}}$ that is black-box in the class group. The following theorem then follows from the HSS and NIZK schemes of [ADOS22] and Theorems 9, 11 and 15.

Corollary 19. *Assuming the hardness of DDH, DXDH, the weak hidden order assumption and small exponent assumption over class groups, there exist*

- *2-server MveHSS scheme with strong privacy for* NC^1 *in the generic bilinear group and random oracle model with local and public knowledge soundness that is black-box in the class group,*
- *2-server MveHSS scheme with strong privacy for SIMD* NC^1 *with non-adaptive local and public soundness in the random oracle model, additionally assuming the hardness of the subgroup decision assumption over composite order bilinear groups, that is black-box in the class group and the composite order bilinear group.*

Additionally, both MveHSS schemes above have a one-round protocol that realizes $\mathcal{F}_{\mathsf{pkgen}}$ *assuming the servers have a common reference string.*

7 Applications

In this section we use HSS with verifiable evaluation for private delegation of computation and for collaborative computation of zkSNARKs.

7.1 Private Delegation of Computation

Private delegation of computation [GGP10] allows a computationally weak client to outsource the computation of a program on private inputs to one or more servers such that the servers do not learn any information about the client's inputs. The servers return the program output and prove the correctness of the returned output to the client. Verification of the proof should require significantly lower computational effort on behalf of the client compared to evaluating the program from scratch. Moreover, the client should obtain a publicly verifiable proof attesting that the program, when evaluated on some private inputs, produces the claimed output.

A ve-HSS scheme provides a natural approach for private delegation of computation as shown in Figure 15. Specifically, given a common reference string computed using a setup process, the client can simply share its private inputs among the servers, who then compute their share of the output and reconstruct it back towards the client. The client then learns the function output as well as a proof of correctness for the output that can be non-interactively verified. The ve-HSS scheme also provides the client with a publicly verifiable non-interactive proof that allows the client to prove that the output is consistent with a public part of the program input.

From Corollaries 13 and 14, we have private delegation of computation schemes for NC^1, SIMD NC^1, P/poly, and SIMD P/poly from a wide range of assumptions that ensure privacy as long as at least one

Protocol $\Pi_{\text{pdc}}^{\text{ve-HSS}}$

The protocol runs with a client P^{c} and m servers $\{P_1, \ldots, P_m\}$ where the client wishes to evaluate a circuit C on public input $\mathbf{x}_{\text{pub}}$ and private input $\mathbf{x}$. Let ve-HSS be an m-server ve-HSS scheme for a class of circuits $\mathbb{C}$ such that $\mathsf{C} \in \mathbb{C}$. The protocol runs in the presence of an adversary $\mathcal{A}$ that corrupts a subset of servers $\mathcal{I} \subset [1, m]$.

Setup: Output $\mathsf{crs} \leftarrow \text{ve-HSS.Setup}(1^\lambda, \mathsf{C})$.

Input Phase: The client P^{c} computes $\langle \mathbf{x} \rangle \leftarrow \text{ve-HSS.Share}(\mathsf{crs}, \mathbf{x})$ and sends $(\mathbf{x}_{\text{pub}}, \langle \mathbf{x} \rangle_i)$ to the i-th server P_i for each $i \in [1, m]$.

Evaluation Phase: Each server P_i computes $[\text{out}]_i \coloneqq \text{ve-HSS.Eval}(\mathsf{crs}, \mathbf{x}_{\text{pub}}, \langle \mathbf{x} \rangle_i)$ where $i \in [1, m]$.

Output Phase: Each server P_i sends $[\text{out}]_i$ to the client P^{c}. P^{c} then computes

$$(\mathbf{y}, \pi_{\text{loc}}, \pi_{\text{pub}}) \coloneqq \text{ve-HSS.Recon}(\mathsf{crs}, [\text{out}]_1, \ldots, [\text{out}]_m).$$

It outputs $\perp$ if either $\text{ve-HSS.LVerify}(\mathsf{crs}, \mathbf{x}_{\text{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\text{loc}}) = 0$ or $\text{ve-HSS.PVerify}(\mathsf{crs}, \mathbf{x}_{\text{pub}}, \mathbf{y}, \pi_{\text{pub}}) = 0$, else it outputs $(\mathbf{x}_{\text{pub}}, \mathbf{y}, \pi_{\text{pub}})$.

Figure 15: Protocol for private delegation of computation using a ve-HSS scheme.

server is honest. An attractive feature of these protocols is that the client is guaranteed the correctness of the output even when all servers are corrupt. Next, we briefly discuss the security and efficiency properties of the protocol.

Security. Correctness of the ve-HSS scheme implies that the local and public proofs verify and that the client obtains the correct circuit output when servers follow the description of the protocol. Privacy of the scheme ensures that an adversary corrupting any subset of the servers, or one that learns the publicly verifiable proof and the circuit output does not gain any additional information about the private inputs of the client. Local soundness of the ve-HSS scheme guarantees the correctness of the circuit output when the local proof verifies successfully, even if an adversary corrupts all servers. Similarly, public soundness ensures the consistency of the circuit output with the public inputs when the public proof verifies successfully, irrespective of the behavior of the servers or the client.

Efficiency. Efficiency of the ve-HSS scheme ensures that the computational effort of the client required for the input phase and verification of output is sublinear in the size of the circuit being computed. Moreover, the resulting protocol is a 2 round protocol for verifiable computation.

7.2 Collaborative zkSNARKs

A zkSNARK for a relation $\mathcal{R}$ allows a prover holding a private witness $\mathbf{w}$ for a statement $\mathbf{x}$ to compute a publicly verifiable proof π that convinces a verifier that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, while ensuring that π does not leak any information about $\mathbf{w}$. Collaborative zkSNARKs [OB22] generalize zkSNARKs from considering a single prover holding a witness $\mathbf{w}$ to the setting where the witness $\mathbf{w} = (w_1, \ldots, w_n)$ is distributed among n provers, where the i-th prover holds w_i. The provers compute a publicly verifiable proof π to prove that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ whilst ensuring that their witness remains private even when π is made public.

We define the ideal functionality for collaborative zkSNARKs in Figure 16. The ideal functionality returns the zkSNARK proof to provers only if they input a valid witness for the statement. If the witness is

Functionality $\mathcal{F}_{\text{czk}}^{\text{SNARK}}$

The functionality $\mathcal{F}_{\text{czk}}^{\text{SNARK}}$ is parameterized by a zkSNARK SNARK and proceeds as follows with n provers in the presence of an adversary Sim that corrupts a subset $\mathcal{I}_c \subset [1, n]$ of provers. Each prover has a private witness w_i and all provers have $(\mathbf{x}, \mathcal{R}, \text{crs})$ as common input, where $\mathbf{x}$ is the public statement, $\mathcal{R}$ is a relation, and crs is a CRS for SNARK.

1: It receives the CRS crs and a relation $\mathcal{R}$ from the honest provers and sends crs to the adversary Sim.

2: It receives the statement $\mathbf{x}$ from the honest provers and sends it to Sim.

3: It receives w_i for every $i \in [1, n] \setminus \mathcal{I}$ from the i-th prover and receives $\{w_i\}_{i \in \mathcal{I}}$ from Sim. Let $\mathbf{w} = (w_1, \ldots, w_n)$.

4: If $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$, it sends $\perp$ to the Sim and all provers. Else, it computes $\pi \leftarrow \text{SNARK.Prove}(\text{crs}, \mathbf{x}, \mathbf{w})$ and sends π to Sim.

5: It receives $\mathcal{S}$ from Sim. For every $i \in [1, n]$, it sends $\perp$ to the i-th client if $i \in \mathcal{S}$ else it sends $(\mathbf{x}, \pi)$ to it.

Figure 16: Ideal functionality for collaborative zkSNARKs.

not valid, the adversary does not learn any additional information about the witness of honest provers. On the other hand, when provers input a valid witness, the ideal functionality does not provide any additional information to the prover than what is leaked by the zkSNARK proof. However, since the proof is zero-knowledge, it preserves the privacy of its witness.

The protocol described in Figure 17 allows a set of provers, each holding a private witness for a public statement, to collaboratively compute a zkSNARK by delegating computation to a set of servers. The adversary can maliciously corrupt provers and a subset of servers. As in our MveHSS construction, the protocol uses an HSS scheme to compute shares of the output of the non-linear phase of the zkSNARK and then leverages linearity of the secret sharing scheme to compute shares of the proof. While the protocol uses the same approach as our MveHSS construction, the construction is simpler in this case since we no longer need to compute the local proofs. On the other hand, the protocol needs to achieve a slightly stronger security requirement where provers obtain the (public) proof only if they hold a valid witness. We achieve this conditional reconstruction of the proof by exploiting the split prover property of the Splittable zkSNARK. Specifically, servers compute an additive sharing of $y \cdot M_{\text{NL}}$ using HSS where $y = \text{C}_{\mathcal{R}}(\mathbf{x}, \mathbf{w})$ denotes the output of the relation circuit on the public statement and the witness input by provers. The servers then compute their share of the proof as $[y \cdot \pi] = \text{Prove}_{\text{L}}([y \cdot M_{\text{NL}}])$. However, before reconstructing the proof, the servers employ a MAC-check to ensure that the output y of the relation-checking circuit $\text{C}_{\mathcal{R}}$ was indeed computed using the witness shared by provers. If the MAC-check passes successfully, the servers publicly reconstruct the shares of the proof. As in the case of our MveHSS construction, we ensure that the randomness used for computing the proof is uniformly distributed, when at least one prover is honest, by deriving it using the randomness shared by each prover.

Observe that once the provers send their inputs to the servers, the protocol allows servers to publicly reconstruct the proof, without requiring any further involvement from the provers. This can serve as a useful property in practical applications where provers can share their witness to delegate computation of the zkSNARK proof and then go offline. The servers can put up their shares of the proof on a public bulletin board to allow any external verifier to reconstruct and verify the validity of the statement.

Theorem 20. *Let* HSS *be an m-server public-key HSS scheme for $\mathbb{C}_{\text{hss}}$ with semi-malicious correctness and let* SNARK *be a $\mathbb{C}_{\text{hss}}$-simple Splittable zkSNARK for $\mathbb{C}$-circuit satisfiability. Then Π_{czk} (Figure 17) realizes $\mathcal{F}_{\text{czk}}^{\text{SNARK}}$ (Figure 16) in the $\{\mathcal{F}_{\text{NIZK}}, \mathcal{F}_{\text{pkgen}}\}$-hybrid model for any relation $\mathcal{R}$, where the relation circuit $\text{C}_{\mathcal{R}} \in \mathbb{C}$ satisfies*

Protocol Π_{czk}

Π_{czk} is described in the $\{\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{pkgen}}\}$-hybrid (Figures 9 and 12) and runs with provers $\mathcal{P}_{\mathsf{c}} = \{P_1^{\mathsf{c}}, \ldots, P_n^{\mathsf{c}}\}$ and servers $\mathcal{P} = \{P_1, \ldots, P_m\}$ in the presence of a static adversary $\mathcal{A}$ that corrupts a subset of servers $\mathcal{I} \subset [1, m]$ and provers $\mathcal{I}_{\mathsf{c}} \subset [1, n]$. Each prover P_i^{c} has private witness w_i and all provers have the public statement $\mathbf{x}$ and a CRS crs.

Public parameters. Let HSS be an m-server public-key HSS scheme that supports evaluating circuits over a finite commutative ring R and let SNARK be a Splittable zkSNARK. Let ℓ_{rand} denote the number of random elements from R required by the zkSNARK prover. Let $\mathcal{R}$ be an NP relation.

Notation. Let $\mathsf{C}_{\mathcal{R}}$ be the relation-checking circuit for $\mathcal{R}$. Let C_{NL} denote the circuit that computes the non-linear phase $\mathsf{Prove}_{\mathsf{NL}}$ of the Splittable zkSNARK for the circuit satisfiability statement $(\mathsf{C}_{\mathcal{R}}, \mathbf{x}, y) \in \mathsf{SAT}$. Let $\widehat{\mathsf{C}}_{\mathsf{NL}}$ denote the circuit that evaluates C_{NL} using randomness derived from the sum of the last n inputs and outputs $\mathsf{C}_{\mathcal{R}}(\mathbf{x}, \mathbf{w}) \cdot \mathsf{C}_{\mathsf{NL}}(\mathbf{x}, \mathbf{w})$, for $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. Let $\widehat{\mathsf{C}}$ be a circuit such that $(\Gamma_y^1, \ldots, \Gamma_y^m) = \widehat{\mathsf{C}}(\mathbf{x}, \mathbf{w}, K_1, \ldots, K_m)$, where $y = \mathsf{C}_{\mathcal{R}}(\mathbf{x}, \mathbf{w})$ and $\Gamma_{\mathsf{y}}^i = y \cdot K_i$ for each $i \in [1, m]$.

1: Each prover and server invokes $\mathcal{F}_{\mathsf{pkgen}}^{\mathsf{HSS}}$ to receive pk and the i-th server P_i additionally receives its evaluation key ek_i.

2: The servers and provers do the following.

1. Each server P_i samples $K_i \leftarrow R$ uniformly at random, computes $\widetilde{K}_i \leftarrow \mathsf{HSS.Enc}(\mathsf{pk}, K_i)$ and a proof of correctness $\pi_{K_i}^{\mathsf{inp}}$ for the statement $(\mathsf{pk}, R, \widetilde{K}_i)$ using $\mathcal{F}_{\mathsf{NIZK}}$.
2. Each prover P_i^{c} samples $\mathbf{r}_i \leftarrow R^{\ell_{\mathsf{rand}}}$ uniformly at random. It then computes $\widetilde{\mathbf{r}}_i \leftarrow \mathsf{HSS.Enc}(\mathsf{pk}, \mathbf{r}_i)$ and $\widetilde{w}_i \leftarrow \mathsf{HSS.Enc}(\mathsf{pk}, w_i)$, and proofs of correct encryption $\pi_{\mathbf{r}_i}^{\mathsf{inp}}$ and $\pi_{w_i}^{\mathsf{inp}}$ using $\mathcal{F}_{\mathsf{NIZK}}$.
3. Each server P_i broadcasts $(\widetilde{K}_i, \pi_{K_i}^{\mathsf{inp}})$ to all servers and each prover P_i^{c} broadcasts $(\mathsf{crs}, \mathbf{x}, \widetilde{\mathbf{r}}_i, \widetilde{w}_i, \pi_{\mathbf{r}_i}^{\mathsf{inp}})$ to all servers.

3: Each server P_i uses $\mathcal{F}_{\mathsf{NIZK}}$ to verify the proofs sent by all provers and servers, and aborts if any proof does not verify. Moreover, if the CRS or statement sent by all provers is not identical it aborts. Else, it computes the following.

$$\left(\left[\Gamma_y^1\right]_i, \ldots, \left[\Gamma_y^m\right]_i\right) \coloneqq \mathsf{HSS.Eval}\left(\mathsf{ek}_i, \widehat{\mathsf{C}}, \mathbf{x}, \widetilde{w}_1, \ldots, \widetilde{w}_n, \widetilde{K}_1, \ldots, \widetilde{K}_n\right),$$
$$[y \cdot M_{\mathsf{NL}}]_i \coloneqq \mathsf{HSS.Eval}\left(\mathsf{ek}_i, \widehat{\mathsf{C}}_{\mathsf{NL}}, \mathbf{x}, \widetilde{w}_1, \ldots, \widetilde{w}_n, \widetilde{\mathbf{r}}_1, \ldots, \widetilde{\mathbf{r}}_n\right),$$
$$[y \cdot \pi]_i \coloneqq \mathsf{SNARK.Prove}_{\mathsf{L}}(\mathsf{crs}, [y \cdot M_{\mathsf{NL}}]_i).$$

For each $j \in [1, m]$, it sends $\left[\Gamma_y^j\right]_i$ to the j-th server P_j.

4: Each server P_i receives $\left[\Gamma_y^j\right]$ and reconstructs it compute Γ_y^i. It then checks if $\Gamma_y^i = K_i$; if so, it broadcasts $[y \cdot \pi]_i$, else it aborts.

5: Provers reconstruct $[y \cdot \pi]$ to either obtain 0 or the proof π. If they obtain 0 or if $\mathsf{SNARK.Verify}(\mathsf{crs}, \mathbf{x}, \pi) = 0$ then they output $\perp$, else they output π.

Figure 17: Collaborative zkSNARK protocol for Splittable zkSNARKs using a public-key HSS scheme.

$\widehat{\mathsf{C}}, \widehat{\mathsf{C}}_{\mathsf{NL}} \in \mathbb{C}_{\mathsf{hss}}$ *for* $\widehat{\mathsf{C}}$ *and* $\widehat{\mathsf{C}}_{\mathsf{NL}}$ *defined in Figure 17. Additionally,* Π_{czk} *is black-box in* HSS *and the linear phase of* SNARK.

Proof Sketch. The proof proceeds similar to that of Theorem 15 and we only discuss the key changes to the simulator here. The simulator works by first simulating the inputs of honest provers, relying on the security of the HSS scheme, and uses $\mathcal{F}_{\mathsf{NIZK}}$ to extract the inputs of corrupt provers. It then sends the extracted inputs to the ideal functionality and obtains the output. It also computes the expected output shares of corrupt servers, which subsequently allows it to verify if the MAC check passes for honest servers. If the ideal functionality returns $\perp$, it reverse samples the shares of the honest servers so that the output reconstructs to 0. Else, it reverse samples the shares of the honest servers so that the output reconstructs to the proof π returned by the ideal functionality. Finally, upon receiving the share sent by the corrupt servers, it can reconstruct the output, using the reverse sampled shares of the honest servers, and check if the reconstructed proof verifies since it knows the statement. ■

Similar to Corollary 19, it follows that the Groth16 zkSNARK can be computed collaboratively for NC^1 relations in the random oracle model. Moreover, if provers are semi-honest, we no longer require $\mathcal{F}_{\mathsf{NIZK}}$ and the semi-malicious correctness of the HSS scheme (see Section 6.2), and it follows from Corollary 17 that the Groth16 zkSNARK can be computed collaboratively for NC^1 and P/poly from a much wider range of assumptions.

Acknowledgements

A majority of this work was done while the first and second authors were at NTT Research. The third and fourth authors were supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. The fourth author was additionally supported in part by JP Morgan Faculty Award, and research gifts from Ethereum, Stellar and Cisco.

References

[ACGS23] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. Cryptology ePrint Archive, Report 2023/1949, 2023. 5

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 421–452, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. 1, 3, 4, 5, 6, 7, 8, 17, 18, 19, 22, 23, 48, 55, 62, 63

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 67–97, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. 7

[BBC+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press. 6

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017. 8

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. 8

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2105–2122, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. 1, 8

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. 1, 8

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press. 1

[BCH+22] Nir Bitansky, Arka Rai Choudhuri, Justin Holmgren, Chethan Kamath, Alex Lombardi, Omer Paneth, and Ron D. Rothblum. PPAD is as hard as LWE and iterated squaring. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part II*,

volume 13748 of *Lecture Notes in Computer Science*, pages 593–622, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland. 12

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333, Tokyo, Japan, March 3–6, 2013. Springer, Berlin, Heidelberg, Germany. 8, 12, 27

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196, Amalfi, Italy, September 3–5, 2014. Springer, Cham, Switzerland. 54

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Heidelberg, Germany. 55

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 1–20, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Heidelberg, Germany. 23

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367, Sofia, Bulgaria, April 26–30, 2015. Springer, Berlin, Heidelberg, Germany. 3, 4, 24, 48, 62

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Berlin, Heidelberg, Germany. 1, 8, 20, 21, 22, 23, 48, 50

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1292–1303, Vienna, Austria, October 24–28, 2016. ACM Press. 6

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 163–193, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland. 1, 8

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94, pages 21:1–21:21, Cambridge, MA, USA, January 11–14, 2018. LIPIcs. 3, 4, 24

[BGIN19] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 869–886, London, UK, November 11–15, 2019. ACM Press. 7

[BGIN20] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 244–276, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland. 7

[BGIN21] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear GMW-style compiler for MPC with preprocessing. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 457–485, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. 7

[BGMM20] James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. Reusable two-round MPC from DDH. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 320–348, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland. 1

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341, Cambridge, MA, USA, February 10–12, 2005. Springer, Berlin, Heidelberg, Germany. 14, 34

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 474–482, Montreal, QC, Canada, June 19–23, 2017. ACM Press. 4

[BKP+23] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. Cryptology ePrint Archive, Report 2023/754, 2023. 4

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 3–33, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland. 1, 3, 4, 7, 8, 22, 23, 48, 62

[BWW23] Eli Bradley, Brent Waters, and David J. Wu. Batch arguments to NIZKs from one-way functions. Cryptology ePrint Archive, Report 2023/1938, 2023. 4

[CGH+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 34–64, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. 7

[CGJ+23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084

of *Lecture Notes in Computer Science*, pages 635–668, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. 4, 14

[Che23] Xin Chen. Verifiable homomorphic secret sharing for machine learning classifiers. *IEEE Access*, 11:43639–43647, 2023. 6, 7

[CHK+19a] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1103–1114, Phoenix, AZ, USA, June 23–26, 2019. ACM Press. 12

[CHK+19b] Arka Rai Choudhuri, Pavel Hubacek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via iterated squaring modulo a composite. Cryptology ePrint Archive, Report 2019/667, 2019. 12

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 394–423, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. 4, 14

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd Annual Symposium on Foundations of Computer Science*, pages 68–79, Denver, CO, USA, February 7–10, 2022. IEEE Computer Society Press. 4, 14, 37

[CL15] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. Cryptology ePrint Archive, Report 2015/047, 2015. 23

[Cle90] Richard Cleve. Towards optimal simulations of formulas by bounded-width programs. In *22nd Annual ACM Symposium on Theory of Computing*, pages 271–277, Baltimore, MD, USA, May 14–16, 1990. ACM Press. 22

[CLMZ23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zkSNARK provers. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 6453–6469, Anaheim, CA, USA, August 9–11, 2023. USENIX Association. 5, 6, 18, 19

[CM21] Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 842–870, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland. 1, 7, 8

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 194–224, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland. 1

[COS+22] Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, Nigel P. Smart, and Barry Van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on FHE. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22: 13th International Conference on Security in Communication Networks*, volume 13409 of *Lecture Notes in Computer Science*, pages 540–563, Amalfi, Italy, September 12–14, 2022. Springer, Cham, Switzerland. 8, 24, 48, 62

[Cou19] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 473–503, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland. 1

[CW23] Jeffrey Champion and David J. Wu. Non-interactive zero-knowledge from non-interactive batch arguments. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 38–71, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. 4

[CZ20] Xin Chen and Liang Feng Zhang. Two-server verifiable homomorphic secret sharing for high-degree polynomials. In Willy Susilo, Robert H. Deng, Fuchun Guo, Yannan Li, and Rolly Intan, editors, *ISC 2020: 23rd International Conference on Information Security*, volume 12472 of *Lecture Notes in Computer Science*, pages 75–91, Bali, Indonesia, December 16–18, 2020. Springer, Cham, Switzerland. 6, 7

[CZ21] Xin Chen and Liang Feng Zhang. Two-server delegation of computation on label-encrypted data. *IEEE Trans. Cloud Comput.*, 9(4):1645–1656, 2021. 6, 7

[dCP22] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 150–179, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland. 6

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 54–74, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Berlin, Heidelberg, Germany. 8

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Berlin, Heidelberg, Germany. 1, 3, 4, 24

[DIJL23] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 315–348, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. 1, 7, 8, 24

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer, Berlin, Heidelberg, Germany. 23

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Heidelberg, Germany. 7

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. 12

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017: 11th International Conference on Provable Security*, volume 10592 of *Lecture Notes in Computer Science*, pages 381–399, Xi'an, China, October 23–25, 2017. Springer, Cham, Switzerland. 8

[FIKW22] Ingerid Fosli, Yuval Ishai, Victor I. Kolobov, and Mary Wootters. On the download rate of homomorphic secret sharing. In Mark Braverman, editor, *ITCS 2022: 13th Innovations in Theoretical Computer Science Conference*, volume 215, pages 71:1–71:22, Berkeley, CA, USA, January 31 – February 3, 2022. LIPIcs. 8

[GGJ+23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 4427–4444, Anaheim, CA, USA, August 9–11, 2023. USENIX Association. 5, 6, 18, 19

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Heidelberg, Germany. 63

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer, Berlin, Heidelberg, Germany. 8, 12, 30

[GGW24] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 449–487, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. 5, 6, 18, 19

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658, Copenhagen, Denmark, May 11–15, 2014. Springer, Berlin, Heidelberg, Germany. 1

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 495–504, New York, NY, USA, May 31 – June 3, 2014. ACM Press. 7

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. 7

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press. 7, 10

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340, Singapore, December 5–9, 2010. Springer, Berlin, Heidelberg, Germany. 8

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Berlin, Heidelberg, Germany. 2, 3, 4, 8, 12, 30, 31, 32, 33

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. 8

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 520–549, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland. 4, 14

[HZ20] Yan He and Liang Feng Zhang. Cheater-identifiable homomorphic secret sharing for outsourcing computations. *J. Ambient Intell. Humaniz. Comput.*, 11(11):5103–5113, 2020. 6

[ILM21] Yuval Ishai, Russell W. F. Lai, and Giulio Malavolta. A geometric approach to homomorphic secret sharing. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 92–119, Virtual Event, May 10–13, 2021. Springer, Cham, Switzerland. 8

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 708–721, Virtual Event, Italy, June 21–25, 2021. ACM Press. 12

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 723–732, Victoria, BC, Canada, May 4–6, 1992. ACM Press. 7

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1115–1124, Phoenix, AZ, USA, June 23–26, 2019. ACM Press. 4

[KPY20] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and PPAD-hardness. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 652–673, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. 12

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 169–189, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Berlin, Heidelberg, Germany. 8

[LMS18] Russell W. F. Lai, Giulio Malavolta, and Dominique Schröder. Homomorphic secret sharing for low degree polynomials. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 279–309, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland. 8

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Heidelberg, Germany. 10

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 632–651, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. 12

[LZW+24] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. Scalable collaborative zk-SNARK: Fully distributed proof generation and malicious security. Cryptology ePrint Archive, Report 2024/143, 2024. 5, 6, 18, 19

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. 2, 7, 8

[MTG22] Arup Mondal, Pratyush Ranjan Tiwari, and Debayan Gupta. Poster: Fully homomorphic secret sharing with output verifiability. *NDSS Poster*, 2022. 6

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 4291–4308, Boston, MA, USA, August 10–12, 2022. USENIX Association. 5, 6, 13, 19, 64

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland. 1, 3, 4, 5, 6, 7, 8, 18, 22, 23, 48, 62

[RRR21] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. *SIAM J. Comput.*, 50(3), 2021. 9, 11

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. 1, 3, 4, 5, 6, 7, 8, 18, 22, 23, 48, 55, 62, 63

[TBM20] Georgia Tsaloli, Gustavo Banegas, and Aikaterini Mitrokotsa. Practical and provably secure distributed aggregation: Verifiable additive homomorphic secret sharing. *Cryptogr.*, 4(3):25, 2020. 6

[TLM18] Georgia Tsaloli, Bei Liang, and Aikaterini Mitrokotsa. Verifiable homomorphic secret sharing. In Joonsang Baek, Willy Susilo, and Jongkil Kim, editors, *ProvSec 2018: 12th International Conference on Provable Security*, volume 11192 of *Lecture Notes in Computer Science*, pages 40–55, Jeju, South Korea, October 25–28, 2018. Springer, Cham, Switzerland. 6

[TM19] Georgia Tsaloli and Aikaterini Mitrokotsa. Sum it up: Verifiable additive homomorphic secret sharing. In Jae Hong Seo, editor, *ICISC 19: 22nd International Conference on Information Security and Cryptology*, volume 11975 of *Lecture Notes in Computer Science*, pages 115–132, Seoul, Korea, December 4–6, 2019. Springer, Cham, Switzerland. 6

[vzGS91] Joachim von zur Gathen and Gadiel Seroussi. Boolean circuits versus arithmetic circuits. *Inf. Comput.*, 91(1):142–154, mar 1991. 50

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 433–463, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. 3, 4, 12, 14, 34, 35, 36

[YO19] Maki Yoshida and Satoshi Obana. Verifiably multiplicative secret sharing. *IEEE Trans. Inf. Theory*, 65(5):3233–3245, 2019. 6, 7

[ZW22] Liang Feng Zhang and Huaxiong Wang. Multi-server verifiable computation of low-degree polynomials. In *2022 IEEE Symposium on Security and Privacy*, pages 596–613, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press. 6, 7