Foundations of Homomorphic Secret Sharing*

Elette Boyle[†] IDC Herzliya
Niv Gilboa[‡] Ben-Gurion University
Yuval Ishai[§] Technion
Huijia Lin[¶] UC Santa Barbara
Stefano Tessaro[‖] UC Santa Barbara

December 27, 2017

Abstract

Homomorphic secret sharing (HSS) is the secret sharing analogue of homomorphic encryption. An HSS scheme supports a local evaluation of functions on shares of one or more secret inputs, such that the resulting shares of the output are short. Some applications require the stronger notion of *additive* HSS, where the shares of the output add up to the output over a finite Abelian group. While strong feasibility results for HSS are known under specific cryptographic assumptions, many natural questions remain open.

We initiate a systematic study of HSS, making the following contributions.

- **A definitional framework.** We present a general framework for defining HSS schemes that unifies and extends several previous notions from the literature, and cast known results within this framework.
- **Limitations.** We establish limitations on *information-theoretic* multi-input HSS with short output shares via a relation with communication complexity. We also show that *additive* HSS for non-trivial functions, even the AND of two input bits, implies non-interactive key exchange, and is therefore unlikely to be implied by public-key encryption or even oblivious transfer.
- **Applications.** We present two types of applications of HSS. First, we construct 2-round protocols for secure multiparty computation from a simple constant-size instance of HSS. As a corollary, we obtain 2-round protocols with attractive asymptotic efficiency features under the Decision Diffie Hellman (DDH) assumption. Second, we use HSS to obtain nearly optimal worst-case to average-case reductions in P. This in turn has applications to fine-grained average-case hardness and verifiable computation.

Keywords: Cryptography, homomorphic secret sharing, secure computation, communication complexity, worst-case to average-case reductions.

*This is a full version of [BGI+18].

[†]Email: eboyle@alum.mit.edu
[‡]Email: gilboan@bgu.ac.il
[§]Email: yuvali@cs.technion.ac.il
[¶]Email: rachel.lin@cs.ucsb.edu
[‖]Email: tessaro@cs.ucsb.edu

1 Introduction

Fully homomorphic encryption (FHE) [RAD78, Gen09] is a powerful cryptographic primitive that supports general computations on encrypted inputs. Despite intensive study, FHE schemes can only be based on a narrow class of cryptographic assumptions [vDGHV10, BV14, GSW13], which are all related to lattices, and their concrete efficiency leaves much to be desired.

In this paper we consider the following secret sharing analogue of FHE, referred to as *homomorphic secret sharing* (HSS) [BGI16a]. A standard (threshold) secret sharing scheme randomly splits an input x into m shares, $(x^1, \ldots, x^m)$, such that any set of t shares reveals nothing about the input. An HSS scheme supports computations on shared inputs by means of local computations on their shares. More concretely, there is a local evaluation algorithm Eval and decoder algorithm Dec satisfying the following homomorphism requirement. Given a description of a function F, the algorithm $\mathsf{Eval}(F; x^j)$ maps an input share x^j to a corresponding output share y^j, such that $\mathsf{Dec}(y^1, \ldots, y^m) = F(x)$.

An HSS scheme as above can be trivially obtained by letting Eval output (F, x^j) and Dec first reconstruct x from the shares and then compute F. However, this trivial scheme is not useful. Analogously to the output compactness requirement of FHE, we require that the HSS output shares be *compact* in the sense that their length depends only on the output length of F and the security parameter. In fact, it is often useful to make the more stringent requirement that Dec compute $F(x)$ as the sum $y^1 + \ldots + y^m$ in some finite Abelian group. We refer to such an HSS scheme as an *additive HSS*. We also consider a relaxed notion of *weak compactness* that allows the length of the output shares to grow sublinearly with the input size.

Finally, one can naturally consider a *multi-input* variant of HSS, where inputs $x_1, \ldots, x_n$ are independently shared, Eval locally maps the j-th shares of the n inputs to the j-th output share, and Dec outputs $F(x_1, \ldots, x_n)$. In fact, multi-input HSS is meaningful even when F is a fixed function rather than an input of Eval. For instance, one may consider additive 2-input HSS where F computes the AND of two input bits, or compact 2-input HSS where F takes an inner product of two input vectors.

HSS vs. FHE. HSS can generally be viewed as a relaxation of FHE that offers protection against bounded collusions. However, as observed in [BGI16a], in some applications of FHE it is possible to use HSS as an alternative that offers the same level of security. For instance, in the context of secure two-party computation [Yao86, GMW87], using HSS to share the inputs of the two parties does not compromise security in any way, since the two parties together can anyway learn both inputs.

More importantly for this work, HSS can potentially offer several useful features that are inherently impossible for FHE. One such feature is *information-theoretic security*. Information-theoretic HSS schemes for multiplying two secrets with security threshold $t < m/2$ serve as the basis for information-theoretic protocols for secure multiparty computation [BGW88, CCD88, CDM00]. Information-theoretic HSS schemes for certain classes of depth-2 circuits implicitly serve as the basis for the best known constructions of information-theoretic private information retrieval schemes and locally decodable codes [Yek07, Efr09, BIKO12]. Another potential feature of HSS is *optimal compactness*: if F has a single output bit, then the output shares y^j can be as short as a single bit. Indeed, special types of FHE schemes [LTV12, CM15, MW16] can be used to obtain additive HSS schemes with $t = m - 1$ that support general homomorphic computations with optimal compactness [DHRW16]. This feature is useful for several applications of HSS, including ones we discuss in

this work.

Finally, recent works obtain HSS schemes that support rich classes of computations under the Decision Diffie Hellman (DDH) assumption [BGI16a, BGI17, BCG+17] or the security of the Paillier encryption scheme [FGJS17, Cou17], which are not known to imply FHE. These constructions use very different techniques from those underlying known FHE constructions. This suggests a potential for further diversifying the assumptions and structures on which HSS can be based, which may potentially lead to more efficient substitutes for known FHE schemes.

1.1 Our Contribution

The current state of the art in HSS mostly consists of isolated positive results and leaves open some of the most basic questions. In this work we initiate a more systematic study of HSS, making the following contributions. We refer the reader to the relevant sections for a high level overview of the main ideas behind each contribution.

A definitional framework. We start, in Section 2, by presenting a general framework for HSS that unifies and extends several previous notions from the literature. In Section 3 we cast some known primitives and previous results within this framework. This includes a simple extension of a previous Learning With Errors (LWE)-based construction from [DHRW16] to the setting of multi-input HSS, whose details appear in Appendix B.

Limitations. In Section 4 we establish two types of limitations on multi-input HSS. First, in Section 4.1, we show that weakly compact *information-theoretic* multi-input HSS schemes for security threshold $t \geq m/2$ do not exist for functions that have high (randomized, one-way) two-party communication complexity. This includes simple functions such as inner product or set disjointness. The high level idea is to obtain a low-communication two-party protocol from the HSS scheme by having the two parties use a common source of randomness to locally simulate the HSS input shares of both inputs, without any interaction, and then have one party send its HSS output share to the other. Second, in Section 4.2, we show that *additive* HSS for non-trivial functions, or even for computing the AND of two input bits, implies non-interactive key exchange (NIKE), a cryptographic primitive which is not known to be implied by standard public-key primitives such as oblivious transfer. Loosely, two parties can simultaneously exchange HSS shares of input bits whose AND is zero, and output their HSS-evaluated output share as a shared key. This result provides some explanation for the difficulty of constructing strong types of HSS schemes from general assumptions.

Applications. In Section 5 we present two types of applications of HSS. First, in Section 5.1, we construct 2-round protocols for secure multiparty computation (MPC) from a simple constant-size instance of additive HSS with $n = 3$ inputs and $m = 2$ shares, for computing $\mathsf{3Mult}(x_1, x_2, x_3) = x_1x_2x_3$. At a very high level, this reduction crucially relies on a randomized encoding of functions by degree-3 polynomials [AIK05], which allows decomposing the computation of an arbitrary function F into the computation of many degree-3 monomials. The computation of each monomial is then performed using many invocation of HSS for $\mathsf{3Mult}$. As a corollary, we can transform a previous DDH-based 2-round MPC protocol in [BGI17] (which requires a public-key infrastructure) for only a constant number of parties, into a 2-round protocol for an arbitrary polynomial number of parties.

In the literature, 2-round MPC protocols exist *in the CRS model*, based on LWE (e.g., [AJLA+12, MW16]) and *in the plain model,* from indistinguishability obfuscation or witness encryption, to-

gether with NIZK (e.g., [GGHR14, GLS15]) or bilinear groups [GS17a]. Very recently, it was shown that 2-round MPC can be based on the (minimal) assumption that 2-round semi-honest oblivious transfer protocols exist [GS17b, BL17]. Our protocol can be instantiated in the public-key infrastructure model under DDH, which is weaker than or incomparable to the feasibility results of other recent constructions. However, our protocols using HSS still have several advantages, in particular, they enjoy better asymptotic efficiency, and they are in the more general "client-server" model, where the computation of the input clients can be done offline, and the computation of the output clients is relatively cheap.

A second type of applications, presented in Section 5.2, is to obtaining worst-case to average-case reductions in P. Roughly speaking, the HSS evaluation function Eval for computing F defines a function $\hat{F}$ such that computing F on any given input x can be reduced to computing $\hat{F}$ on two or more inputs that are individually pseudorandom. A similar application of FHE was already pointed out in [CKV10]. However, an advantage of the HSS-based reductions is that they allow $\hat{F}$ to have a single bit of output. Another advantage is the potential for diversifying assumptions. We discuss applications of the reductions implied by HSS to fine-grained average-case hardness and verifiable computation. In particular, the HSS-based approach yields checking procedures for polynomial-time computations that achieve better soundness vs. succinctness tradeoffs than any other approach we are aware of.

2 General Definitional Framework for HSS

In this section we give a general definition of homomorphic secret sharing (HSS) that can be instantiated to capture different notions from the literature. We consider multi-input HSS schemes that support a compact evaluation of a function F on shares of inputs $x_1, \ldots, x_n$ that originate from different clients. More concretely, each client i randomly splits its input x_i between m servers using the algorithm Share, so that x_i is hidden from any t colluding servers. We assume $t = m - 1$ by default. Each server j applies a local evaluation algorithm Eval to its share of the n inputs, and obtains an output share y^j. The output $F(x_1, \ldots, x_n)$ is reconstructed by applying a decoding algorithm Dec to the output shares $(y^1, \ldots, y^m)$.

As discussed in the Introduction, HSS as above can be trivially realized by letting Share be any m-out-of-m secret sharing scheme, Eval the identity function, and Dec a function that reconstructs the inputs from their shares and then applies F. However, applications of HSS require that Dec be in some sense "simpler" than computing F. The most natural simplicity requirement, referred to as *compactness*, is that the output length of Eval, and hence the complexity of Dec, depend only on the output length of F and not on the input length of F. A more useful notion of simplicity is the stronger requirement of *additive* decoding, where the decoder computes the exclusive-or of the output shares or, more generally, adds them up in some Abelian group $\mathbb{G}$. We also consider weaker notions of simplicity that are motivated by applications of HSS and are needed to capture HSS constructions from the literature.

Finally, for some of the main applications of HSS it is useful to let F and Eval take an additional input x_0 that is known to all servers. This is necessary for a meaningful notion of single-input HSS (with $n = 1$). Typically, the input x_0 will be a description of a function f applied to the input of a single client, e.g., a description of a circuit, branching program, or low-degree polynomial. The case of single-input HSS is considerably different from the case of multi-input HSS with no server input. In particular, the negative results presented in this work do not apply to single-input HSS.

We now give our formal definition of general HSS. We refer the reader to Example 3.1 for an example of using this definition to describe an HSS scheme for multiplying two field elements using Shamir's secret sharing scheme. Here and in the following, we use the notation $\Pr[A_1; \ldots; A_m : E]$ to denote the probability that event E occurs following an experiment defined by executing the sequence $A_1, \ldots, A_m$ in order.

Definition 2.1 (HSS)**.** An n-client, m-server, t-secure homomorphic secret sharing scheme for a function $F : (\{0,1\}^*)^{n+1} \to \{0,1\}^*$, or (n, m, t)-*HSS* for short, is a triple of PPT algorithms $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Share}(1^\lambda, i, x)$: On input 1^λ (security parameter), $i \in [n]$ (client index), and $x \in \{0,1\}^*$ (client input), the sharing algorithm Share outputs m input shares, $(x^1, \ldots, x^m)$. By default, we require Share to run in (probabilistic) polynomial time in its input length; however, we also consider a relaxed notion of efficiency where Share is given the total length ℓ of *all* $n+1$ inputs (including x_0) and may run in time $\mathsf{poly}(\lambda, \ell)$.
- $\mathsf{Eval}\big(j, x_0, (x_1^j, \ldots, x_n^j)\big)$: On input $j \in [m]$ (server index), $x_0 \in \{0,1\}^*$ (common server input), and $x_1^j, \ldots, x_n^j$ (jth share of each client input), the evaluation algorithm Eval outputs $y^j \in \{0,1\}^*$, corresponding to server j's share of $F(x_0; x_1, \ldots, x_n)$.
- $\mathsf{Dec}(y^1, \ldots, y^m)$: On input $(y^1, \ldots, y^m)$ (list of output shares), the decoding algorithm Dec computes a final output $y \in \{0,1\}^*$.

The algorithms $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ should satisfy the following correctness and security requirements:

- **Correctness:** For any $n+1$ inputs $x_0, \ldots, x_n \in \{0,1\}^*$,

$$\Pr\left[\begin{array}{l} \forall i \in [n]\ (x_i^1, \ldots, x_i^m) \leftarrow \mathsf{Share}(1^\lambda, i, x_i) \\ \forall j \in [m]\ y^j \leftarrow \mathsf{Eval}\big(j, x_0, (x_1^j, \ldots, x_n^j)\big) \end{array} : \mathsf{Dec}(y^1, \ldots, y^m) = F(x_0; x_1, \ldots, x_n)\right] = 1.$$

  Alternatively, in a *statistically correct* HSS the above probability is at least $1 - \mu(\lambda)$ for some negligible μ and in a δ-correct HSS (or δ-HSS for short) it is at least $1 - \delta - \mu(\lambda)$. In the case of δ-HSS the error parameter δ may be given as an additional input to Eval, and the running time of Eval is allowed to grow polynomially with $1/\delta$.

- **Security:** Consider the following semantic security challenge experiment for corrupted set of servers $T \subset [m]$:
  1: The adversary gives challenge index and inputs $(i, x, x') \leftarrow \mathcal{A}(1^\lambda)$, with $i \in [n]$ and $|x| = |x'|$.
  2: The challenger samples $b \leftarrow \{0,1\}$ and $(x^1, \ldots, x^m) \leftarrow \mathsf{Share}(1^\lambda, i, \tilde{x})$, where $\tilde{x} = \begin{cases} x \text{ if } b = 0 \\ x' \text{ else} \end{cases}$.
  3: The adversary outputs a guess $b' \leftarrow \mathcal{A}((x^j)_{j \in T})$, given the shares for corrupted T.

  Denote by $\mathsf{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ the advantage of $\mathcal{A}$ in guessing b in the above experiment, where probability is taken over the randomness of the challenger and of $\mathcal{A}$. For circuit size bound $S = S(\lambda)$ and advantage bound $\alpha = \alpha(\lambda)$, we say that an (n, m, t)-HSS scheme $\Pi = (\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ is (S, α)-secure if for all $T \subset [m]$ of size $|T| \le t$, and all non-uniform adversaries $\mathcal{A}$ of size $S(\lambda)$, we have $\mathsf{Adv}(1^\lambda, \mathcal{A}, T) \le \alpha(\lambda)$. We say that Π is:

– *computationally secure* if it is $(S, 1/S)$-secure for all polynomials S;
– *statistically α-secure* if it is (S, α)-secure for all S;
– *statistically secure* if it statistically α-secure for some negligible $\alpha(\lambda)$;
– *perfectly secure* if it is statistically 0-secure.

Remark 2.2 (Unbounded HSS)**.** Definition 2.1 treats the number of inputs n as being fixed. We can naturally consider an unbounded multi-input variant of HSS where F is defined over arbitrary sequences of inputs x_i, and the correctness requirement is extended accordingly. We denote this flavor of multi-input HSS by $(*, m, t)$-HSS. More generally, one can allow all three parameters n, m, t to be flexible, treating them as inputs of the three algorithms $\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec}$.

Remark 2.3 (Robust decoding)**.** Definition 2.1 allows Dec to use all output shares for decoding the output. When $t < m - 1$, one can consider a stronger variant of HSS where Dec can recover the output from any $t + 1$ output shares. Such a robust notion of threshold homomorphic encryption was recently considered in [JRS17]. In this work we do not consider robust decoding.

Remark 2.4 (Functions vs. relations)**.** In this work we only consider HSS for *functions,* where the output is fully determined by the inputs. It is sometimes useful to consider the more general problem of HSS for *relations* (or search problems), where there may be more than one admissible output. The notion of share conversion from [BIKO12] can be seen as a special case of HSS for relations.

2.1 Notions of Simple Decoding

As discussed above, Definition 2.1 can be trivially realized by Eval that computes the identity function. To make HSS useful, we impose two types of requirements on the decoding algorithm. The strictest requirement is that Dec simply add the output shares over a finite field $\mathbb{F}$ or, more generally, an Abelian group $\mathbb{G}$. We refer to such an HSS scheme as being *additive*, and assume by default (unless a group or field are specified) that Dec computes the exclusive-or of its inputs, namely the group is of the form $\mathbb{G} = \mathbb{Z}_2^\ell$. Note that any HSS scheme where Dec computes a fixed linear combination of the output shares (over some finite field or ring) can be converted into an additive scheme by letting Eval multiply its outputs by the coefficients of the linear combination. See Example 3.1 for a relevant concrete example.

A more liberal requirement is *compactness*, which says that the length of the output shares depends only on the output length and the security parameter, independently of the input length. Finally, we also consider a further relaxation that we call *weak compactness*, requiring that the length of the output shares be sublinear in the input length when the input length is sufficiently bigger than the security parameter and the output length. See Remark 2.7 for motivation and further discussion. We formalize these notions below, and then discuss other variants of additive and compact HSS in Remarks 2.6 and 2.7.

Definition 2.5 (Additive and compact HSS)**.** We say that an (n, m, t)-HSS scheme $\Pi = (\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ for F is:

- *Additive* if Dec outputs the exclusive-or of the m output shares, or $\mathbb{G}$-additive if Dec computes addition in an Abelian group $\mathbb{G}$ (see Remark 2.6 for further generalizations);

- *Compact* if there is a polynomial p such that for every $\lambda, \ell_{\mathsf{out}}$, and inputs $x_0, x_1, \ldots, x_n \in \{0,1\}^*$ such that $|F(x_0; x_1, \ldots, x_n)| = \ell_{\mathsf{out}}$, the length of each output share obtained by applying Share with security parameter λ and then Eval is at most $p(\lambda) \cdot \ell_{\mathsf{out}}$ (or $O(\ell_{\mathsf{out}})$ for perfect or statistically α-secure HSS with a constant α);

- *Weakly compact* if there is a polynomial p and sublinear function $g(\ell) = o(\ell)$, such that for every $\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}}$, and inputs $x_0, x_1, \ldots, x_n \in \{0,1\}^*$ of total length ℓ_{in} with $|F(x_0; x_1, \ldots, x_n)| = \ell_{\mathsf{out}}$, the length of each output share obtained by applying Share with security parameter λ and then Eval is at most $g(\ell_{\mathsf{in}}) + p(\lambda) \cdot \ell_{\mathsf{out}}$ (or $g(\ell_{\mathsf{in}}) + O(\ell_{\mathsf{out}})$ for perfect or statistically α-secure HSS with a constant α). More generally, we can specify the precise level of compactness by referring to an HSS scheme as being $g(\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}})$-compact.

Remark 2.6 (Generalized additive HSS)**.** Our default notion of additive HSS considers the output of F as an element in a group of the form $\mathbb{Z}_2^\ell$, namely Dec computes the bitwise exclusive-or of its inputs. Some applications motivate additive HSS with respect to a general Abelian group $\mathbb{G}$ that can be given as an additional input. In particular, this is useful for applications that require aggregating outputs on different inputs (see, e.g., [BGI15, BGI16a]). The above definition can be extended to accommodate this by allowing F to take a description of $\mathbb{G}$ as an additional input, where $F(\mathbb{G}, x_0; x_1, \ldots, x_n)$ is guaranteed to take values from $\mathbb{G}$, and allowing all of the HSS algorithms to receive $\mathbb{G}$ as an additional input. In this case Dec performs addition over the group $\mathbb{G}$ given as input. In another dimension of generalization, it is may be useful to require that the m outputs jointly form a valid encoding of the output of F with respect to a given linear code. This is useful for the robust reconstruction feature discussed in Remark 2.3.

Remark 2.7 (On the different notions of compactness)**.** The main notion of compactness from Definition 2.5 corresponds to the strict notion of compactness that is typically required in the context of FHE. It is easy to see that every additive HSS is also compact in this sense. However, even in the case of perfect HSS, there are natural constructions that are compact but are not additive. Such HSS schemes are implicit in the best known constructions of information-theoretic private information retrieval schemes and locally decodable codes [Yek07, Efr09, BIKO12]. The more liberal notion of weak compactness serves two purposes. First, our negative results for information-theoretic HSS apply to this notion, which makes them stronger. Second, the liberal notion is needed to capture some known HSS schemes in which the length of the output shares needs to grow (sublinearly) with the input length; see Section 3.

Finally, another variant of the compactness requirement imposes a restriction on the *computational complexity* of Dec. Note that our default notion of compactness implies that Dec is polynomial in ℓ_{out} and λ, but one may consider relaxed variants that allow sublinear dependence on the input length. See Section 5.2 for applications that depend on the computational complexity of Dec.

2.2 Default Conventions

It is convenient to make the following default choices of parameters and other conventions.

- We assume $t = m - 1$ by default and write (n, m)-HSS for $(n, m, m-1)$-HSS.

- We assume computational security by default, and refer to the statistical and perfect variants collectively as "information-theoretic HSS."

- In the case of *perfectly secure* or *statistically α-secure* HSS, λ is omitted.
- For $n \geq 2$ clients, we assume by default that the servers have no input and write $F(x_1, \ldots, x_n)$, omitting the server input x_0. Note that (n, m, t)-HSS with server input can be reduced to $(n+1, m, t)$-HSS with no server input by letting the server input be shared by one of the clients.
- We consider *additive HSS* by default. This stronger notion is useful for several application of HSS, and most HSS constructions realize it.
- We will sometimes be interested in additive HSS for a constant-size (finite) function F, such as the AND of two bits; this can be cast into Definition 2.1 by just considering an extension $\hat{F}$ of F that outputs 0 on all invalid inputs. Note that our two notions of compactness are not meaningful for a constant-size F. We can similarly handle functions F that impose restrictions on the relation between the lengths of different inputs. Since Eval can know all inputs lengths, we can ensure that Dec output 0 in case of mismatch.
- As noted above, the common server input x_0 is often interpreted as a "program" P from a class of programs $\mathcal{P}$ (e.g., circuits or branching programs), and F is the universal function defined by $F(P; x_1, \ldots, x_n) = P(x_1, \ldots, x_n)$. We refer to this as *HSS for the class $\mathcal{P}$*.

2.3 HSS with Setup

When considering multi-input HSS schemes, known constructions require different forms of *setup* to coordinate between clients. This setup is generated by a PPT algorithm Setup and is reusable, in the sense that the same setup can be used to share an arbitrary number of inputs. We consider the following types of setup:

- *No setup*: This is the default notion of HSS defined above.
- *Common random string (CRS) setup*: An algorithm $\mathsf{Setup}(1^\lambda)$ is used to generate a uniformly random string σ which is given as input to Share, Eval, and Dec. The correctness and security definitions are extended in the natural way, where both the adversary and the challenger in the security game are given access to an honestly generated σ. This can be relaxed to a common *reference* string that can be picked from an arbitrary distribution. However, this relaxation will not be useful in this work.
- *Public-key setup:* We consider here a strong form of public-key setup in which $\mathsf{Setup}(1^\lambda)$ outputs a public key pk and m secret evaluation keys $(\mathsf{ek}_1, \ldots, \mathsf{ek}_m)$, where each key is given to a different server. The algorithm Share is given pk as an additional input, and $\mathsf{Eval}\big(j, \ldots\big)$ is given ek_j as an additional input. The security game is changed by giving both the adversary and the challenger pk and giving to the adversary $(\mathsf{ek}_j)_{j \in T}$ in addition to $(x^j)_{j \in T}$. Following the terminology from [BGI17], we refer to HSS with this type of setup as *public-key $(*, m, t)$-HSS*.

3 Constructions

In this section we present positive results on HSS that are either implicit in the literature or can be easily obtained from known results. We cast these results in terms of the general HSS framework

from Section 2.

We start with a detailed example for casting Shamir's secret sharing scheme [Sha79] over a finite field $\mathbb{F}$ as a perfectly secure, $\mathbb{F}$-additive $(2, m, t)$-HSS scheme for the function F that multiplies two field elements. Such a scheme exists if and only if $m > 2t$.

Example 3.1 (Additive $(2, m, t)$-HSS for field multiplication)**.** *Let m, t be parameters such that $m > 2t$, let $\mathbb{F}$ be a finite field with $|\mathbb{F}| > m$, let $\theta_1, \ldots, \theta_m$ be distinct nonzero field elements, and let $\lambda_1, \ldots, \lambda_n$ be field elements ("Lagrange coefficients") such that for any univariate polynomial p over $\mathbb{F}$ of degree at most $2t$ we have $p(0) = \sum_{j=1}^{m} \lambda_j p(\theta_j)$. Let $F : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ be the (constant-size) function defined by $F(x_1, x_2) = x_1 \cdot x_2$. A perfectly secure, additive $(2, m, t)$-HSS scheme for F is defined by the following algorithms. (Since F is a constant-size function we are not concerned with efficiency; we also omit x_0 since there is no server input and omit the security parameter λ since security is perfect.)*

1. $\mathsf{Share}(i, x)$*: pick $r_1, \ldots, r_t$ uniformly at random from $\mathbb{F}$ and let $p(Z) = x + r_1 Z + r_2 Z^2 + \ldots + r_t Z^t$ be a random polynomial of degree at most t with x as its free coefficient. Output $(p(\theta_1), \ldots, p(\theta_m))$. Note that* Share *does not depend on i (the inputs are shared in the same way).*

2. $\mathsf{Eval}\big(j, (x_1^j, x_2^j)\big)$*: Output $\lambda_j \cdot x_1^j x_2^j$.*

3. $\mathsf{Dec}(y^1, \ldots, y^m)$*: Output $y^1 + \ldots + y^m$.*

We now survey some other instances of HSS schemes from the literature.

- Additive m-out-of-m secret sharing over an Abelian group $\mathbb{G}$ is a $\mathbb{G}$-additive, perfectly secure $(*, m)$-HSS for the function $F(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ where $x_i \in \mathbb{G}$. This is the first instance of HSS considered in the literature [Ben86].

- Generalizing Example 3.1, multiplicative secret sharing [CDM00] over a finite field $\mathbb{F}$ is an $\mathbb{F}$-additive, perfectly secure $(2, m, t)$-HSS for the function F that multiplies two field elements. Such schemes exist if and only if $m > 2t$. Multiplicative secret sharing schemes such as Shamir's scheme serve as the basis for secure multiparty computation protocols in the information-theoretic setting [BGW88, CCD88]. More generally, information-theoretic $\mathbb{F}$-additive (d, m, t)-HSS for multiplying d elements of $\mathbb{F}$ exists if and only if $m > dt$ [BIW10]. Multiplicative schemes with a smaller threshold t that work over a constant-size field (independent of m) can be based on algebraic geometric codes [CC06]. Efficient multiplicative schemes that support a pointwise multiplication of two vectors are considered in [FY92, CCCX09].

- A 1-round k-server private information retrieval (PIR) scheme [CGKS98, CG97] can be seen as a *weakly compact* $(1, k, 1)$-HSS for the selection function $F(D; \gamma) = D_\gamma$. For the 2-server case ($k = 2$), information theoretic PIR schemes provably cannot achieve our stronger notion of compactness unless the share size is linear in $|D|$ [GKST06, KdW04]. Moreover, current schemes only realize our relaxed notion of efficiency for Share, since the share size is super-polynomial in $|\gamma|$ (see [DG15] for the best known construction in terms of total size of input shares and output shares). In the computational case, there are in fact *additive* 2-server schemes based on the existence of one-way functions, where Share satisfies the default strict notion of efficiency (see [BGI16b] for the best known construction).

- Non-trivial instances of compact, perfectly-secure (1,3,1)-HSS for certain classes of depth-2 boolean circuits [BIKO12] implicitly serve as the basis for the best known constructions of information-theoretic 3-server PIR schemes and 3-query locally decodable codes [Yek07, Efr09].

- The main result of [BGI16a] is a construction of (single-input, computationally secure, additive) $(1, 2)$-δ-HSS for branching programs under the DDH assumption. The same paper also obtains a public-key $(*, 2)$-δ-HSS variant of this result. Similar results assuming the circular security of the Paillier encryption were recently obtained in [FGJS17, Cou17].

- The notion of function secret sharing (FSS) from [BGI15] is dual to the notion of HSS for a program class $\mathcal{P}$. It can be cast as an additive $(1, m)$-HSS for the universal function $F(x; P) = P(x)$, where $P \in \mathcal{P}$ is a program given as input to the client and x is the common server input. The special case of distributed point function (DPF) [GI14] is FSS for the class of point functions (namely, functions that have nonzero output for at most one input). DPF can be seen as additive $(1, m)$-HSS for the function $F(x; (\alpha, \beta))$ that outputs β if $x = \alpha$ and outputs 0 otherwise. It is known that one-way functions are necessary and sufficient for DPF with $m = 2$ servers [GI14]. Whether they are sufficient for 3-server DPF is open.

- We observe that additive[1] $(*, m)$-HSS for *circuits* with statistical correctness can be obtained from the Learning With Errors (LWE) assumption, by a simple variation of the FSS construction from spooky encryption of [DHRW16] (more specifically, their techniques for obtaining 2-round MPC). The share size in this construction must grow with the circuit depth, hence Share only satisfies the relaxed notion of efficiency; this dependence can be eliminated by relying on a stronger variant of LWE that involves circular security. We provide details of the underlying tools and construction in Appendix B.

We note that a key feature of HSS is that Dec does not require a secret key. This rules out nontrivial instances of single-server HSS. In particular, single-server PIR [KO97] and fully homomorphic encryption [Gen09] cannot be cast as instances of our general definitional framework of HSS.

4 Limitations

In this section, we discuss some inherent limitations in HSS. First, in Section 4.1, we show lower bounds on the length of output shares in statistically-secure HSS using communication complexity lower bounds. Then, in Section 4.2, we show that additive $(2, 2)$-HSS for the AND of two bits implies non-interactive key-exchange (NIKE). Given what is known about NIKE (in particular it only follows from non-generic assumptions, and it is not known to be implied directly by public-key encryption or OT), this gives a strong justification for the lack of instantiations from generic assumptions.

[1] If one does not insist on additive HSS and settles for the weaker notion of compactness, then single-input HSS can be trivially obtained from any FHE scheme by letting Share include an encryption of the input in one of the shares and split the decryption key into m shares.

4.1 Lower Bounds for Statistically-Secure Multi-Input HSS

We show lower bounds on the length of output shares in statistically-secure multi-input HSS using communication-complexity lower bounds. The key step is to derive a public-coin two-party protocol to compute a function F from an HSS scheme for the function F, with the additional property that the communication cost of the resulting protocol only depends on the length of the *output* shares.

4.1.1 Communication Complexity Refresher

We consider an arbitrary public-coin interactive protocol Π between two parties, Alice and Bob, who start the execution with respective inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, and common random tape R. (We can assume wlog that the protocol is otherwise deterministic, and all random coins come from R. Moreover, the sets $\mathcal{X}$ and $\mathcal{Y}$ are finite here.) At any point in the execution, one of the parties can return an output value, denoted $\Pi(R, x, y)$. The cost of Π is the maximum number of bits exchanged by Alice and Bob, taken as the worst case over all possible inputs x, y, and random tapes R. We also say that such a protocol is *one-way* (or *one round*) if only one message is sent, and this goes from Alice to Bob.

We are interested in the inherent cost of a protocol Π that evaluates a function $F : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$. In particular, the *(randomized) communication complexity of F with error ϵ*, denoted $\mathsf{R}_\epsilon(F)$, is the minimum cost of a public-coin protocol Π such that $\Pr[\Pi(R, x, y) \neq F(x, y)] \leq \epsilon$ for all x, y, where the probability is over the public random string R. If we restrict ourselves to one-way protocols, then we define analogously the *one-way communication complexity of F with error ϵ*, denoted $\mathsf{R}_\epsilon^{A \to B}(F)$. It is clear that $R_\epsilon^{A \to B}(F) \geq R_\epsilon(F)$ must always hold.

The following are classical examples of lower bounds on the (one-way) randomized communication complexity. (We note that the choice of the constant $1/3$ is arbitrary, as any other constant $< \frac{1}{2}$ would do.)

Theorem 4.1 (e.g., [KN97])**.** *Let $\mathsf{IP}_\ell : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be such that $\mathsf{IP}_\ell(x, y) = \sum_{i=1}^{\ell} x_i y_i \pmod 2$. Then, $\mathsf{R}_{1/3}(\mathsf{IP}_\ell) = \Omega(\ell)$.*

Theorem 4.2 ([KS92])**.** *Let $\mathsf{DISJ}_\ell : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be such that $\mathsf{DISJ}_\ell(x, y) = \neg \bigvee_{i=1}^{\ell} (x_i \wedge y_i)$. Then, $\mathsf{R}_{1/3}(\mathsf{DISJ}_\ell) = \Omega(\ell)$.*

Theorem 4.3 ([KNR99])**.** *Let $\mathsf{INDEX}_\ell : \{0,1\}^\ell \times [\ell] \to \{0,1\}$ be such that $\mathsf{INDEX}_\ell(x_1 x_2 \ldots x_\ell, i) = x_i$. Then, $\mathsf{R}_{1/3}^{A \to B}(\mathsf{INDEX}_\ell) = \Omega(\ell)$.*

4.1.2 Lower Bounds on the Length of Output Shares

We start with a lower bound on the length of the output shares in $(2, 2)$-HSS. (Recall that (n, m)-HSS is a shorthand for $(n, m, t = m - 1)$-HSS.) Below, we describe informally how to extend the technique to more general settings.

Recall that a $(2, 2)$-HSS scheme is defined for a function $F : (\{0,1\}^*)^2 \to \{0,1\}^*$. (In this section, we consider the case where the servers have no input x_0, but the results extend straightforwardly to handle server inputs.) For any two integers $\ell_{1,\mathsf{in}}, \ell_{2,\mathsf{in}}$, it is convenient to define the restriction $F^{\ell_{1,\mathsf{in}}, \ell_{2,\mathsf{in}}} : \{0,1\}^{\ell_{1,\mathsf{in}}} \times \{0,1\}^{\ell_{2,\mathsf{in}}} \to \{0,1\}^*$ such that $F^{\ell_{1,\mathsf{in}}, \ell_{2,\mathsf{in}}}(x_1, x_2) = F(x_1, x_2)$. Also, for a suitable function g, we say that a $(2, 2)$-HSS scheme is *g-compact* if, for security parameter λ, when the two inputs have lengths $\ell_{1,\mathsf{in}}$ and $\ell_{2,\mathsf{in}}$, respectively, the output shares have length each at most $g(\lambda, \ell_{1,\mathsf{in}}, \ell_{2,\mathsf{in}})$.

Proposition 4.4 (Compactness lower bound)**.** *Let* $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ *be a* $(2,2)$*-HSS scheme for a function* $F : (\{0,1\}^*)^2 \to \{0,1\}^*$*, which is statistically* α*-secure,* g*-compact, and* δ*-correct. Then, for all* λ*, and* $\ell_{1,\mathsf{in}}, \ell_{2,\mathsf{in}} > 0$*,*

$$g(\lambda, \ell_{1,\mathsf{in}}, \ell_{2,\mathsf{in}}) \geq \mathsf{R}^{A\to B}_{\delta(\lambda)+4\alpha(\lambda)}(F^{\ell_{1,\mathsf{in}},\ell_{2,\mathsf{in}}}) \; . \tag{1}$$

Proof. Let $\alpha = \alpha(\lambda)$, $\delta = \delta(\lambda)$. We first consider the following private-coin protocol Π to compute the function $F^{\ell_{1,\mathsf{in}},\ell_{2,\mathsf{in}}}$ for all $x_1 \in \{0,1\}^{\ell_{1,\mathsf{in}}}$ and $x_2 \in \{0,1\}^{\ell_{2,\mathsf{in}}}$:

Protocol $\Pi(x_1, x_2)$:

- Alice runs $(x_1^1, x_1^2) \leftarrow \mathsf{Share}(1^\lambda, 1, x_1)$. Bob runs $(x_2^1, x_2^2) \leftarrow \mathsf{Share}(1^\lambda, 2, x_2)$.
- Bob sends x_2^1 to Alice.
- Alice computes $y_1 \leftarrow \mathsf{Eval}(1^\lambda, 1, (x_1^1, x_2^1))$, and sends (x_1^2, y_1) to Bob.
- Bob computes $y_2 \leftarrow \mathsf{Eval}(1^\lambda, 2, (x_1^2, x_2^2))$, $y \leftarrow \mathsf{Dec}(1^\lambda, y_1, y_2)$, and outputs y.

Clearly, the protocol has error probability at most δ. We now show how to use public randomness and reverse sampling, as well as HSS security, to eliminate the need to exchange the input shares x_2^1 and x_1^2. To this end, let us define the following distributions on pairs of input shares (x^1, x^2):

- Distribution $D_i(x)$: $(x^1, x^2) \leftarrow \mathsf{Share}(1^\lambda, i, x)$ for $i = 1, 2$
- Distribution $\widetilde{D}_1(x)$: $(\widetilde{x}^1, x^2) \leftarrow \mathsf{Share}(1^\lambda, 1, 0^{\ell_{1,\mathsf{in}}})$, $x^1 \leftarrow D_1(x)|x^2$, where $D_1(x)|x^2$ is the distribution of x^1 in $D_1(x)$ conditioned on the second input-share value being equal x^2 (if it is well defined), and an arbitrary distribution otherwise.
- Distribution $\widetilde{D}_2(x)$: $(x^1, \widetilde{x}^2) \leftarrow \mathsf{Share}(1^\lambda, 2, 0^{\ell_{2,\mathsf{in}}})$, $x^2 \leftarrow D_2(x)|x^1$, where $D_2(x)|x^1$ is the distribution of x^2 in $D_2(x)$ conditioned on the second input-share value being equal x^1 (if it is well defined), and an arbitrary distribution otherwise.

The statistical distance ϵ_1 between $D_1(x)$ and $\widetilde{D}_1(x)$ is at most 2α. This is because $D_1(x)$ can equivalently be sampled by replacing $0^{\ell_{1,\mathsf{in}}}$ with x in $\widetilde{D}_1(x)$ and therefore, the statistical distance between $D_1(x)$ and $\widetilde{D}_1(x)$ cannot be larger than the statistical distance between the x^1's sampled in $D_1(x)$ and $\widetilde{D}_1(x)$. Then, by statistical α-security, this distance cannot be larger than 2α. Similarly, the distance ϵ_2 between $D_2(x)$ and $\widetilde{D}_2(x)$ is also at most 2α. This can be exploited to obtain the following one-way public-coin[2] protocol Π' with cost at most $g(\lambda, \ell_{1,\mathsf{in}}, \ell_{2,\mathsf{in}})$.

[2]Here we allow the distribution of the public coins to be arbitrary. One can force them to be uniform by using them as the randomness needed to generate our public-coin distributions.

Public coin generation:

- Sample (x_1^2, x_2^1), where $(\widetilde{x}_1^1, x_1^2) \leftarrow \mathsf{Share}(1^\lambda, 1, 0^{\ell_{1,\mathsf{in}}})$, $(x_2^1, \widetilde{x}_2^2) \leftarrow \mathsf{Share}(1^\lambda, 2, 0^{\ell_{2,\mathsf{in}}})$

Protocol $\Pi'((x_2^1, x_1^2), x_1, x_2)$:

- Alice first samples $x_1^1 \leftarrow D_1(x_1)|x_1^2$, then computes $y_1 \leftarrow \mathsf{Eval}(1^\lambda, 1, (x_1^1, x_2^1))$, and sends y_1 to Bob.
- Bob first samples $x_2^2 \leftarrow D_2(x_2)|x_2^1$, then computes $y_2 \leftarrow \mathsf{Eval}(1^\lambda, 2, (x_1^2, x_2^2))$, and finally outputs $y \leftarrow \mathsf{Dec}(1^\lambda, y_1, y_2)$.

The output of Π' can be seen as a (possibly randomized) function of the four inputs shares $x_1^1, x_1^2, x_2^1, x_2^2$, and similarly, the output of Π can be obtained by applying the same function to the input shares. As the statistical distance between the joint distributions of all four shares in Π' and Π is at most $\epsilon_1 + \epsilon_2 \leq 4\alpha$, the error probability can increase by at most 4α when moving from Π to Π'. Therefore, the length of y_1 in Π' must be at least $\mathsf{R}_{\delta+4\alpha}(F^{\ell_{1,\mathsf{in}},\ell_{2,\mathsf{in}}})$, which concludes the proof. □

As an application, consider any statistically secure $(2,2)$-HSS scheme for inner products, i.e., for the function $\mathsf{IP} : (\{0,1\}^*)^2 \to \{0,1\}$ such that $\mathsf{IP}(x_1, x_2) = \mathsf{IP}_\ell(x_1, x_2)$ whenever $|x_1| = |x_2| = \ell$. Then, the following corollary implies that such scheme cannot be weakly compact. Similar lower bounds can be obtained for disjointness, and for the index function.

Corollary 4.5. *There exists no weakly compact, statistically* $1/24$*-secure* $1/6$*-correct* $(2,2)$*-HSS scheme for* IP*.*

Proof. Apply Proposition 4.4 with $\ell_{1,\mathsf{in}} = \ell_{2,\mathsf{in}} = \ell$, $\delta = 1/6$, and $\alpha = \frac{1}{24}$. Regardless of the security parameter, the length of the output shares must be at least $\mathsf{R}_{1/3}^{A\to B}(\mathsf{IP}_\ell) = \Omega(\ell_{1,\mathsf{in}} + \ell_{2,\mathsf{in}})$ by Theorem 4.1, and this violates weak compactness. □

Extensions. Proposition 4.4 can be extended to obtain lower bounds for general (n, m, t)-HSS where $m, n \geq 2$ and $t \geq m/2$. We briefly summarize the main ideas here.

- $(n, 2)$*-HSS.* For any n-ary function $F : (\{0,1\}^*)^n \to \{0,1\}^*$, we can define a two-party function as follows. Fix $k \in \{1, \ldots, n-1\}$, as well as Alice's indices $I_1 = (a_1, \ldots, a_k)$, and Bob's indices $I_2 = (b_1, \ldots, b_{n-k})$, where $\{a_1, \ldots, a_k, b_1, \ldots, b_{n-k}\} = [n]$. Then,

$$F'((x_{a_1}, \ldots, x_{a_k}), (x_{b_1}, \ldots, x_{b_{n-k}})) = F(x_1, \ldots, x_n) \ .$$

  The proof of Proposition 4.4 can be adapted to lower bound the length of the output shares in an $(n, 2)$-HSS scheme for F via $\mathsf{R}^{A\to B}(F')$, noting one would then choose the sets I_1, I_2 to maximize communication complexity of the resulting F'.

- (n, m, t)*-HSS for* $t \geq m/2$*.* A lower bound for $(n, 2)$-HSS extends straightforwardly to a lower bound for (n, m, t)-HSS where $t \geq m/2$, since the latter type of HSS implies the former type, by simply having one of the two servers in the $(n, 2)$-HSS simulate $m/2 \leq m_1 \leq t$ servers from the (n, m, t)-HSS scheme, and the other simulate the remaining $m_2 = m - m_1$ servers.

- *Simultaneous messages.* In the case of (n, m)-HSS, where $n \geq m$, we can alternatively obtain useful lower bounds via communication complexity in the *simultaneous message model* [KNR99, BGKL03], where m players send a message to a referee that decides the output. Roughly, a variant of the proof of Proposition 4.4 would build a protocol where the messages sent are exactly the m servers' output shares.

4.2 Additive Multi-Input HSS Implies Non-Interactive Key Exchange

It is known that roughly any non-trivial additive HSS (even for a single input) implies the existence of one-way functions [GI14, BGI15]; in turn, one-way functions have been shown to imply (single-input) additive $(1, 2)$- and $(1, m)$-HSS for certain classes of simple functions [CG97, GI14, BGI15, BGI16b]. However, to date, all constructions of additive HSS supporting *multiple* inputs rely on a select list of heavily structured assumptions: DDH, LWE, Paillier, and obfuscation [BGI16a, DHRW16, FGJS17]. A clear challenge is whether one can instantiate such an object from weaker general assumptions, such as one-way functions, public-key encryption, or oblivious transfer.

We show that this is unlikely to occur. We demonstrate the power of additive multi-input HSS by proving that even the minimal version of $(2, 2)$-additive-HSS for the AND of two input bits already implies the existence of *non-interactive key exchange (NIKE)* [DH76], a well-studied cryptographic notion whose known constructions similarly are limited to select structured assumptions. NIKE is black-box separated from one-way functions and highly unlikely to be implied by generic public-key encryption or oblivious transfer.

On the other hand, we observe that $(2, 2)$-additive-HSS for AND is unlikely to be implied *by* NIKE, as the primitive additionally implies the existence of 2-message oblivious transfer (OT) [BGI16a], unknown to follow from NIKE alone.

We first recall the definition of NIKE. For a two-party protocol Π between Alice and Bob, we denote by $\mathsf{out}_A(\Pi)$ and $\mathsf{out}_B(\Pi)$ their respective outputs, and $\mathsf{Transc}(\Pi)$ the resulting transcript.

Definition 4.6 (NIKE)**.** A 2-party protocol Π with single-bit output is a secure *non-interactive key-exchange (NIKE)* protocol if the following conditions hold:

- **Non-Interactive:** The protocol Π consists of exchanging a single (simultaneous) message.
- **Correctness:** The parties agree on a consistent output bit: $\Pr[\mathsf{out}_A(\Pi) = \mathsf{out}_B(\Pi)] = 1$, over randomness of Π.
- **Security:** There exists a negligible function ν such that for any non-uniform polynomial-time E, for every $\lambda \in \mathbb{N}$, it holds $\Pr[b \leftarrow E(1^\lambda, \mathsf{Transc}(\Pi)) : b = \mathsf{out}_A(\Pi)] \leq 1/2 + \nu(\lambda)$, where probability is taken over the randomness of Π and E.

Proposition 4.7. *The existence of additive* $(2, 2)$*-HSS for the AND function* $F : \{0, 1\}^2 \to \{0, 1\}$ *defined by* $F(x_1, x_2) = x_1 x_2$ *implies the existence of non-interactive key exchange.*

Proof. Consider the candidate NIKE protocol given in Figure 1.

Non-interactive: By construction, the protocol consists of a single communication round.

Correctness: Follows by the additive decoding correctness of the $(2, 2)$-HSS for AND. Namely, with probability 1, it holds $z^A + z^B = 0 \in \{0, 1\}$; that is, $z^A = z^B$.

Security: Suppose there exists a polynomial-time eavesdropper E who, given the transcript of the protocol x^B, y^A succeeds in predicting Bob's output bit $z^B = \mathsf{Eval}(B(x^B, y^B))$ with advantage

Communication Round:

- Alice samples shares of 0: i.e., $(x^A, x^B) \leftarrow \mathsf{Share}(1^\lambda, A, 0)$.
  Send x^B to Bob.
- Bob samples a random bit $b \leftarrow \{0,1\}$ and shares b: $(y^A, y^B) \leftarrow \mathsf{Share}(1^\lambda, B, b)$.
  Send y^A to Alice.

Output round:

- Alice outputs $z^A = \mathsf{Eval}(A, (x^A, y^A)) \in \{0,1\}$.
- Bob outputs $z^B = \mathsf{Eval}(B, (x^B, y^B)) \in \{0,1\}$.

Figure 1: NIKE protocol from any additive $(2,2)$-HSS for AND.

α: i.e.,

$$Pr\left[\begin{array}{c} b \leftarrow 0,1; \\ (x^A, x^B) \leftarrow \mathsf{Share}(1^\lambda, A, 0); \\ (y^A, y^B) \leftarrow \mathsf{Share}(1^\lambda, B, b); \\ b' \leftarrow E(1^\lambda, (x^B, y^A)) \end{array} : b' = \mathsf{Eval}(B, (x^B, y^B))\right] \geq 1/2 + \alpha(\lambda).$$

We prove in such case α must be negligible, via the following two claims.

Claim 4.8. *E must succeed with advantage α if Alice shares 1 instead of 0: Explicitly, there exists a negligible function ν_1 for which*

$$Pr\left[\begin{array}{c} b \leftarrow 0,1; \\ (x^A, x^B) \leftarrow \mathsf{Share}(1^\lambda, A, 1); \\ (y^A, y^B) \leftarrow \mathsf{Share}(1^\lambda, B, b); \\ b' \leftarrow E(1^\lambda, (x^B, y^A)) \end{array} : b' = \mathsf{Eval}(B, (x^B, y^B))\right] \geq 1/2 + \alpha(\lambda) - \nu_1(\lambda).$$

Proof of Claim 4.8. Follows by the security of Alice's HSS execution. Namely, consider a distinguishing adversary D for the $(2,2)$-AND-HSS, who performs the following:

1: Sample a random bit $b \leftarrow \{0,1\}$, and HSS share b as $(y^A, y^B) \leftarrow \mathsf{Share}(1^\lambda, B, b)$.
2: Receive a challenge secret share x^B, generated either as $(x^A, x^B) \leftarrow \mathsf{Share}(1^\lambda, A, 0)$ or $(x^A, x^B) \leftarrow \mathsf{Share}(1^\lambda, A, 1)$.
3: Execute E on "transcript" x^B and y^A: Let $b' \leftarrow E(1^\lambda, (x^B, y^A))$.
4: Output 0 if and only if $b' = b$.

By construction, the distinguishing advantage of D is exactly the difference in the prediction advantage of E from the real protocol and the protocol in which Alice shares 1 instead of 0. Thus, this difference must be bounded by some negligible function ν_1. □

Claim 4.9. *The prediction advantage $\alpha(\lambda)$ of E must be negligible in λ.*

Proof of Claim 4.9. Follows by the security of Bob's HSS execution. Namely, consider a distinguishing adversary D for the $(2,2)$-AND-HSS, who performs the following:

1: Generate HSS shares of 1, as $(x^A, x^B) \leftarrow \mathsf{Share}(1^\lambda, A, 1)$.

2: Receive challenge secret share y^A, generated as $(y^A, y^B) \leftarrow \mathsf{Share}(1^\lambda, B, b)$ for random challenge bit $b \leftarrow \{0,1\}$.
3: Execute E on "transcript" x^B and y^A: Let $b' \leftarrow E(1^\lambda, (x^B, y^A))$.
4: Output b' as a guess for b.

By construction, the distinguishing advantage of D is precisely $\alpha(\lambda) - \nu_1(\lambda)$. Thus (since ν_1 is negligible), it must be that α is negligible, as desired. □

This concludes the proof of Proposition 4.7. □

As a direct corollary of this result, any form of HSS which implies additive $(2,2)$-HSS for AND automatically implies NIKE as well. This includes HSS for any functionality F with an embedded AND in its truth table.

As an example, consider a form of *split* distributed point function [GI14], where the nonzero input value $\alpha \in \{0,1\}^\ell$ of the secret point function f_α is held split as additive shares across two clients. This corresponds to additive $(2,2)$-HSS for the function $F(x; \alpha_1, \alpha_2) = [x == (\alpha_1 \oplus \alpha_2)]$ (i.e., evaluates to 1 if and only if $x = \alpha_1 \oplus \alpha_2$). Such a notion would have applications for secure computation protocols involving large public databases, where the index α of the desired data item is not known to either party, but rather determined as the result of an intermediate computation. Unfortunately, we show that such a tool (even for inputs of length 2 bits) implies NIKE, and thus is unlikely to exist from lightweight primitives.

Corollary 4.10. *The existence of "split" DPF, i.e. additive* $(2,2)$*-HSS for the function* $F(x; \alpha_1, \alpha_2) = [x == (\alpha_1 \oplus \alpha_2)]$*, implies the existence of NIKE.*

Proof. Consider the special case of 2-bit values $\alpha_0, \alpha_1 \in \{0,1\}^2$. We show evaluation of F enables evaluation of AND of clients' input bits, and thus additive $(2,2)$-HSS for AND. Indeed, for any $b_1, b_2 \in \{0,1\}$, observe that $F((0,0); (1,b_1), (b_2,1)) = [(0,0) == ((1,b_1) \oplus (b_2,1))] = [(0,0) == (b_1 \oplus 1, b_2 \oplus 1)] = b_1 \wedge b_2$. □

5 Applications

In this section we present two types of applications of HSS. In Section 5.1 we present an application to 2-round secure multiparty computation, and in Section 5.2 we present an application to worst-case to average-case reductions.

5.1 From $(3,2)$-HSS to 2-Round MPC

Let us define the following function over $\mathbb{Z}_2$: $\mathsf{3Mult}(x_1,\ x_2,\ x_3) = x_1 x_2 x_3$. In this section, we show that $(3,2)$-HSS for $\mathsf{3Mult}$ implies 2-round MPC for arbitrary functions in the client-server model. Recall that (n,m)-HSS refers to HSS with n clients, m servers, tolerating up to $m-1$ corrupted servers. Similarly, a (n,m)-MPC protocol is a MPC protocol in the client server model with n clients, m servers, and is semi-honest secure against up to $(m-1)$ corrupted servers.

Theorem 5.1. *Assume the existence of PRGs in* NC^1*. For any* n, m*, and any polynomial-time computable function* $F : (\{0,1\}^*)^n \to \{0,1\}$*, there is a construction of an* (n,m)*-MPC protocol that securely computes* F*, from an additive* $(3,2)$*-HSS for* $\mathsf{3Mult}$*.*

Combining this with the additive δ-HSS construction of [BGI16a] from DDH would result in (n, m)-MPC from DDH with (at best) only $1/\mathsf{poly}(\lambda)$ correctness. Fortunately, we can do better. Indeed, as an intermediate step in the proof of Theorem 5.1 (Lemmas 5.7 and 5.8 below), we prove that $(3, 3)$-MPC for 3Mult-Plus also suffices to imply (n, m)-MPC for general functions. A construction of $(3, 3)$-MPC for general functions (in the PKI model) was shown to follow from DDH in [BGI17] (in fact, they obtain (n, c)-MPC for any constant number of servers c). Combining this with Lemmas 5.7 and 5.8, and the fact that PRGs in NC^1 also follow from DDH, we obtain the following result. This improves directly over the 2-round MPC result of [BGI17], by supporting an arbitrary polynomial number of servers instead of constant. A detailed comparison between our 2-round MPC protocols and other recent ones is provided in Section 5.1.3.

Corollary 5.2 (2-round MPC from DDH)**.** *For any n, m, and any polynomial-time computable function $F : (\{0,1\}^*)^n \to \{0,1\}$, there is a construction of an (n, m)-MPC protocol that securely computes F in the PKI model, assuming DDH.*

Overview of Proof of Theorem 5.1 We prove Theorem 5.1 by combining the following steps.

Step 1: $(3, 2)$-HSS for 3Mult-Plus. Starting from an additive $(3, 2)$-HSS scheme Π_{3Mult} for the function 3Mult, thanks to the property of additive reconstruction, we can directly modify it to obtain an additive $(3, 2)$-HSS for the function 3Mult-Plus (again over $\mathbb{Z}_2$) defined as

$$\mathsf{3Mult\text{-}Plus}((x_1, z_1),\ (x_2, z_2),\ (x_3, z_3)) \quad = \quad x_1x_2x_3 + z_1 + z_2 + z_3\ .$$

Lemma 5.3. *There is a construction of additive $(3, 2)$-HSS for the function 3Mult-Plus from any additive $(3, 2)$-HSS for the function 3Mult.*

Proof. An additive $(3, 2)$-HSS for 3Mult-Plus can be obtained by simply combining the additive $(3, 2)$-HSS Π_{3Mult} for 3Mult with an additive $(3, 2)$-HSS Π_{Add} for addition as follows: The 3 clients and 2 servers run the protocol Π_{3Mult} and Π_{Add} in parallel, using inputs x_1, x_2, x_3 in the former and inputs z_1, z_2, z_3 in the latter, respectively. Each server S_j for $j \in [2]$ obtains two output shares y^j_{3Mult} and y^j_{Add}, and outputs $y^j = y^j_{\mathsf{3Mult}} + y^j_{\mathsf{Add}}$. It follows from the correctness and security of Π_{3Mult} and Π_{Add} that the above described scheme is a correct and secure HSS scheme for $\Pi_{\mathsf{3Mult\text{-}Plus}}$. □

Step 2: $(3, 3)$-MPC for 3Mult-Plus. From an additive $(3, 2)$-HSS scheme for 3Mult-Plus, we can use the *server-emulation technique* from [BGI17] to construct a 3-client *3-server* MPC protocol for 3Mult-Plus. In fact, the technique in [BGI17] is way more general, it shows that from any given n-client m-server HSS for 3Mult-Plus, one can construct a n-client m^2-server MPC protocol for any n-ary function F, assuming the existence of low-depth PRGs.

Lemma 5.4 (Server-Emulation in [BGI17])**.** *Assume existence of PRGs in NC^1. For any n, m and polynomial-time function $F : (\{0,1\}^*)^n \to \{0,1\}$, there is a construction of an (n, m^2)-MPC protocol Π that securely computes F, from an additive (n, m)-HSS for 3Mult-Plus.*

Their general lemma implies the following corollary we need, using the fact that one can reduce the number of servers by having a single server simulating multiple ones. See Claim 5.6.

Corollary 5.5. *Assume the existence of PRGs in NC^1. There is a construction of a $(3, 3)$-MPC protocol that securely computes 3Mult-Plus, from an additive $(3, 2)$-HSS for 3Mult-Plus.*

Claim 5.6. *For any n, m, $m' < m$, and any polynomial-time computable function $F : (\{0,1\}^*)^n \to \{0,1\}$, there is a construction of a (n, m')-MPC protocol Π' that securely computes F, from a (n, m)-MPC protocol Π that securely computes F.*

Proof. To reduce the number of servers, in Π', the first $m'-1$ servers act exactly as the first $m'-1$ servers in Π, and the last server acts as all of the last $m - m' + 1$ servers in Π. Correctness holds trivially. Security holds as the view of an attacker participating in an execution of Π' corrupting a strict subset $\mathcal{S}$ of the m' servers, and an arbitrary subset $\mathcal{T}$ of the clients, is identical to the view of an attacker participating in an execution of Π, corrupting the same set of clients, and still a strict subset $\mathcal{S}'$ of the m servers. In particular $\mathcal{S}' = \mathcal{S}$ if $\mathcal{S}$ does not contain the m''th server, and $\mathcal{S}' = \mathcal{S} \cup \{m', m'+1, \cdots, m\}$ otherwise. Therefore, if Π securely computes F, so does Π'. □

Step 3: $(3, m)$-MPC for 3Mult-Plus — Increase the number of servers. Next, from a $(3,3)$-MPC protocol for computing 3Mult-Plus, we show how to construct $(3, m)$-MPC protocol for computing the same function 3Mult-Plus, with an arbitrary number m of servers.

Lemma 5.7. *For any m, there is a construction of $(3, m)$-MPC protocol that securely computes 3Mult-Plus, from a $(3,3)$-MPC protocol that securely computes 3Mult-Plus.*

Proof Overview. Let Π^3 be a $(3,3)$-MPC protocol for 3Mult-Plus; consider m servers, and three clients C_1, C_2, and C_3. Recall that each client C_d has input (x_d, z_d). If we naively let the three clients execute Π^3 with some subset of 3 servers, in the case all three servers are corrupted, the security of Π^3 no longer holds, and the inputs of all clients are potentially revealed. Thus, the challenge is ensuring that when all but one server is corrupted, the inputs of honest clients remain hidden. To achieve this, each client secret-shares its input bit $x_d = \sum_j s_j^d$; as long as server S_j is uncorrupted, the j'th share s_j^d for each honest client's input x_d remains hidden, and hence so are the inputs x^d. (The additive part z_d of the inputs can be hidden easily; we omit this part in this brief overview.) Towards this, note that multiplying x_1, x_2, x_3 boils down to computing the sum of all possible degree 3 monomials over the shares $x_1x_2x_3 = \sum_{ijk} s_i^1 s_j^2 s_k^3$. Our idea is using the protocol Π^3 to compute each monomial $s_i^1 s_j^2 s_k^3$ hidden with some random blinding bits, and using a protocol Π_{Add} for addition to cancel out these random blinding bits, as well as add z_1, z_2, z_3, all of which done in parallel. More specifically,

- for every i, j, k, C_1, C_2, C_3 together with three appropriate servers described below run Π^3 to enable the output client to obtain $M_{ijk} = s_i^1 s_j^2 s_k^3 + t_{ijk}^1 + t_{ijk}^2 + t_{ijk}^3$, where t_{ijk}^d is a random blinding bit sampled by client C_d;
- in parallel, C_1, C_2, C_3 together with all m servers run a $(3, m)$-MPC protocol Π_{Add} to enable the output client to obtain the sum $T = T^1 + T^2 + T^3$, where $T^d = z_d - \sum_{i,j,k} t_{ijk}^d$;
- finally, the output client adds all M_{ijk} with T, which gives the correct output, i.e., $x_1x_2x_3 + z_1 + z_2 + z_3$.

The only question left is what are the three servers involved for computing M_{ijk}; they naturally should be servers S_i, S_j, S_k, since for an honest client, say C_1, if server S_i is uncorrupted, the share s_i^1 remains hidden in all computations of M_{ijk} involving this share. This allows us to argue security. One technicality is that some monomials may have form $s_i^1 s_i^2 s_j^3$ or $s_i^1 s_i^2 s_i^3$ and only correspond to two servers S_i, S_j or one S_i. In the former case, we will use the $(3,2)$-MPC protocol Π^2, and in the latter case, we directly implement a trivial protocol with one server. □

A full proof is provided in Section 5.1.1.

Step 4: (n, m)-MPC for F — Increase the number of clients and handle general functions. Finally, we show how to construct MPC protocols for computing any n-ary function F, from MPC protocols for computing $\mathsf{3Mult\text{-}Plus}$, using the same number m of servers.

Lemma 5.8. *Assume the existence of PRGs in* NC^1*. For any* n, m*, and any polynomial-time computable function* $F : (\{0,1\}^*)^n \to \{0,1\}$*, there is a construction of* (n, m)*-MPC protocol that securely computes* F*, from a* $(3, m)$*-MPC protocol that securely computes* $\mathsf{3Mult\text{-}Plus}$*.*

Proof Overview. Staring from a $(3, m)$-MPC protocol $\Pi_{\mathsf{3Mult\text{-}Plus}}$ for $\mathsf{3Mult\text{-}Plus}$, our goal is constructing a (n, m)-MPC protocol Π_F for an arbitrary F with an arbitrary number of clients. To do so, we reduce the task of computing F to the task of computing a *degree-3* randomized encoding $\mathsf{RE}_F(x_1, \cdots, x_n;\ r)$ of F. Here, having a degree of 3 means that RE_F can be represented as a degree 3 polynomial in its input *and* random bits. Such a randomized encoding scheme is constructed in [IK02, AIK04], assuming the existence of a low-depth PRG. The first question is where does the random tape r come from. A natural choice is having $r = r_1 + \cdots + r_n$ contributed by all clients. When the randomized encoding has degree 3, its computation can be expanded into a sum of degree three monomials, that is, $\mathsf{RE}_F(x_1, \cdots, x_n\ ;\ r = r_1 + \cdots + r_n) = \sum a^{\ell}_{ijk}\ v_i v_j v_k$, where each variable v_i is either a bit in some input x_l or a bit in some random tape r_l. This decomposes the computation of F into many 3-way multiplications, which can be done securely using $\mathsf{3Mult\text{-}Plus}$. More specifically, in the protocol Π_F,

- for every monomial $a^{\ell}_{ijk} v_i v_j v_k$, the three clients $C_{l_i}, C_{l_j}, C_{l_k}$ holding the variables v_i, v_j, v_k run $\Pi_{\mathsf{3Mult\text{-}Plus}}$ with all m servers to enable the output client to obtain $M_{ijk} = a^{\ell}_{ijk} v_i v_j v_k + t^{\ell,1}_{ijk} + t^{\ell,2}_{ijk} + t^{\ell,3}_{ijk}$, where the three t variables are random blinding bits sampled by the three clients respectively;
- in parallel, all clients and servers run a (n, m)-MPC protocol Π_{Add} for addition to enable the output client to obtain the sum of all t blinding elements;
- the output client adds all M_{ijk} terms, subtracts the sum of blinding elements to obtain the randomized encoding of F, and decodes the randomized encoding.

□

A full proof is provided in Section 5.1.2.

5.1.1 Increase the Number of Servers – Proof of Lemma 5.7

In this section we construct a $(3, m)$-MPC protocol Π^m that securely computes $\mathsf{3Mult\text{-}Plus}$ for any number m of servers. Our construction makes use of the following sub-protocols:

- A $(3, 3)$-MPC protocol Π^3 for computing $\mathsf{3Mult\text{-}Plus}$, and a $(3, 2)$-MPC protocol Π^2 for computing $\mathsf{3Mult\text{-}Plus}$. By Claim 5.6, the latter is implied by the former.
- A $(3, m)$-MPC protocol Π_{Add} for computing Add_3 as constructed in Claim 5.9 below.

Claim 5.9 (MPC for ADD)**.** *For any n, m, there is a construction of (n, m)-MPC protocol that (perfectly) securely computes $\mathsf{Add}_n : \{0,1\}^n \to \{0,1\}$ defined below:*

$$\mathsf{Add}_n(x_1, x_2, \cdots, x_n) = \sum_{i \in [n]} x_i$$

Proof. To compute the sum, each client C_i samples a random m-way secret share of its input x_i, that is, $s_{i1}, \cdots, s_{im}$ such that $\sum_{j \in [m]} s_{ij} = x_i$, and sends share s_{ij} to server S_j for all $j \in [m]$. Each server S_j sums up all the shares it receives as its output share $y_j = \sum_{i \in [n]} s_{ij}$. It is easy to see that correctness holds. For security, if all clients are corrupted, the view of the corrupted parties can be emulated honestly, since all inputs are known to the simulator. Otherwise, since at least one server is uncorrupted, the shares sent from the honest clients to corrupted servers can be emulated using random bits. This emulation is perfect, since without knowing the shares that honest clients send to honest servers, the marginal distribution of the shares sent to corrupted servers are random. □

The Protocol Π^m: The three (input) clients C_1, C_2, C_3 receive respectively inputs (x_1, z_1), (x_2, z_2) and (x_3, z_3); the m servers $S_1, \cdots S_m$ and the output client O have no input. They proceed as follows to compute $\mathsf{3Mult\text{-}Plus}((x_1, z_1),\ (x_2, z_2),\ (x_3, z_3)) = x_1 x_2 x_3 + z_1 + z_2 + z_3$.

1. Each client C_d for $d \in [3]$ additively share its input x_d, by sampling $\{s_i^d\}_{i \in [m]}$ randomly from $\{0,1\}$ subject to $\sum s_i^d = x_d$. C_d additionally sample random bits t_{ijk}^d for every $i, j, k \in [m]$.
2. For every $i, j, k \in [m]$, clients C_1, C_2, C_3 work with servers S_i, S_j, S_k and O to compute
$$M_{ijk} = s_i^1 s_j^2 s_k^3 + t_{ijk}^1 + t_{ijk}^2 + t_{ijk}^3 \ ,$$
by running in parallel appropriate protocols as described below.

   Case 1: i, j, k are all distinct. In this case, C_1, C_2, C_3 execute protocol Π^3 with S_i, S_j, S_k and O, using inputs (s_i^1, t_{ijk}^1), (s_j^2, t_{ijk}^2), and (s_k^3, t_{ijk}^3) respectively.

   Case 2: Exactly two of i, j, k are equal.
   - If they have pattern (i, i, j), the clients execute the protocol Π^2 with S_i, S_j and O, using inputs (s_i^1, t_{iij}^1), (s_i^2, t_{iij}^2), and (s_k^3, t_{iij}^3) respectively.
   - If they have pattern (i, j, i), the clients execute the protocol Π^2 with S_i, S_j and O, using inputs (s_i^1, t_{iji}^1), (s_j^2, t_{iji}^2), and (s_i^3, t_{iji}^3) respectively.
   - If they have pattern (j, i, i), the clients execute the protocol Π^2 with S_i, S_j and O, using inputs (s_j^1, t_{jii}^1), (s_i^2, t_{jii}^2), and (s_i^3, t_{jii}^3) respectively.

   Case 3: If $i = j = k$, each client C_d sends (s_i^d, t_{iii}^d) directly to server S_i, who computes M_{iii} and sends it to O. Denote this simple protocol as Π^1.

   In each case, O obtains M_{ijk}.
3. Clients C_1, C_2, C_3 run Π_{Add} with all servers $S_1, \cdots, S_m$ and O to compute the sum
$$T = \sum_{d \in [3]} T^d, \qquad \text{where } T^d = z_d - \sum_{i,j,k \in [m]} t_{ijk}^d \ ,$$
where C_d uses input T^d, and O obtains output T. This execution is carried out in parallel with all executions in the previous step.

4. Finally, the output client O outputs

$$y = \sum_{i,j,k\in[m]} M_{i,j,k} + T$$

Correctness: It is easy to verify that O obtains the correct output.

$$\begin{aligned} y &= \sum_{i,j,k\in[m]} \left(s_i^1 s_j^2 s_k^3 + t_{ijk}^1 + t_{ijk}^2 + t_{ijk}^3\right) + \sum_{d\in[3]} \left(z_d - \sum_{i,j,k\in[m]} t_{ijk}^d\right) \\ &= \sum_{i,j,k\in[m]} s_i^1 s_j^2 s_k^3 + \sum_{d\in[3]} z_d \\ &= x_1x_2x_3 + z_1 + z_2 + z_3 = \mathsf{3Mult\text{-}Plus}((x_1,z_1),\ (x_2,z_2),\ (x_3,z_3)) \end{aligned}$$

The Simulator Sim^m of Π^m: To show the security of Π^m, we need to construct a simulator Sim^m, such that, for every possible corruption set $\mathcal{I}$ excluding at least one server, Sim^m can simulate the view of the corrupted players using only the inputs of the corrupted clients, and the output if the output client is corrupted. Formally, let $\mathsf{Real}_{\Pi^m}(\mathcal{I}, \{(x_d, z_d)\}_{d\in[3]})$ denote the view of the corrupted parties $\mathcal{I}$ in an execution of Π^m with client inputs $\{(x_d, z_d)\}_{d\in[3]}$. The simulator Sim^m must satisfy:

$$\left\{\mathsf{Real}_{\Pi^m}(\mathcal{I},\ \{x_d,z_d\}_{d\in[3]})\right\}_\lambda \approx \left\{\mathsf{Sim}^m(\mathcal{I},\ \{x_d,z_d\}_{d \text{ s.t. } C_d\in\mathcal{I}},\ y')\right\}_\lambda, \qquad y' = \begin{cases} y & O\in\mathcal{I} \\ \perp & O\notin\mathcal{I} \end{cases}$$

We now formally describe the simulation procedure $\mathsf{Sim}^m(\mathcal{I}, \{(x_d, z_d)\}_{d \text{ s.t. } C_d\in\mathcal{I}}, y)$.

If O is corrupted, Sim^m proceeds as follows:

1. Sim^m samples the following shares and blinding bits.

   Sampling I: For every corrupted client $C_d \in \mathcal{I}$, Sim^m samples $\{s_i^d\}$ at random subject to that $\sum_{i\in[m]} s_i^d = x_d$, and samples t_{ijk}^d at random for every $i,j,k \in [m]$.

   Sampling II: For every $i,j,k \in [m]$, such that, S_i, S_j, S_k are all corrupted, Sim^m samples s_i^1, s_j^2, s_k^3 and $t_{ijk}^1, t_{ijk}^2, t_{ijk}^3$ at random.

   Sampling III: If all clients are corrupted, that is, $\{C_1, C_2, C_3\} \subseteq \mathcal{I}$, compute for all i,j,k

   $$M_{ijk} = s_i^1 s_j^2 s_k^3 + t_{ijk}^1 + t_{ijk}^2 + t_{ijk}^3\ ,$$

   using shares and blinding bits sampled in step Sampling I.

   Otherwise, compute M_{ijk} honestly for i,j,k, such that, S_i, S_j, S_k are all corrupted, using shares and blinding bits sampled in step Sampling II. For every i,j,k, such that, not all of S_i, S_j, S_k are corrupted, sample M_{ijk} at random.

   Let $\mathcal{S}$ denote the set of shares sampled, $\mathcal{T}$ the set of blinding bits, and $\mathcal{M}$ the set of M_{ijk} bits computed or sampled.

2. For every $i,j,k \in [m]$, Sim^m simulates the execution for computing M_{ijk} in two cases depending on whether all servers involved for this computation are corrupted or not.

Case I: $\{S_i, S_j, S_k\} \subseteq \mathcal{I}$. In this case, the security of protocols Π^3, Π^2, and Π^1 no longer hold as all servers are corrupted. Sim^m finds shares s_i^1, s_j^2, s_k^3 and blinding bits t_{ijk}^1, t_{ijk}^2, t_{ijk}^3 in $\mathcal{S}$ and $\mathcal{T}$ respectively (sampled in step Sampling II). It emulates the view of the corrupted parties in the execution for computing M_{ijk} by emulating the honest clients using inputs (s_i^1, t_{ijk}^1), (s_j^2, t_{ijk}^2), (s_k^3, t_{ijk}^3).

Case II: $\{S_i, S_j, S_k\} \not\subseteq \mathcal{I}$. In this case, Sim^m relies on the security of protocols Π^3, Π^2, and Π^1 to simulate.

Case 1: i, j, k are all distinct. Sim^m finds share $s_{i_d}^d$ ($i_d = i$ if $d = 1$, $i_d = j$ if $d = 2$, and $i_d = k$ if $d = 3$) and blinding bit t_{ijk}^d of each corrupted client $C_d \in \mathcal{I}$ in $\mathcal{S}$ and $\mathcal{T}$ respectively (sampled in step Sampling I). It then invokes the simulator Sim^3 of protocol Π^3 with inputs

$$\mathsf{Sim}^3\left(\mathcal{I} \cap \{C_1, C_2, C_3, S_i, S_j, S_k, O\},\ \{s_{i_d}^d, t_{ijk}^d\}_{d \text{ s.t. } C_d \in \mathcal{I}},\ M_{ijk}\right)$$

to simulate the view of the corrupted parties in the execution for computing M_{ijk}.

Case 2: Exactly two of i, j, k are equal. Sim^m finds the relevant share $s_{i_d}^d$ and blinding bit t_{ijk}^d of each corrupted client $C_d \in \mathcal{I}$ in $\mathcal{S}$ and $\mathcal{T}$ as before. It then invokes the simulator Sim^2 of protocol Π^2 with inputs

$$\mathsf{Sim}^3\left(\mathcal{I} \cap \{C_1, C_2, C_3, S_i, S_j, S_k, O\},\ \{s_{i_d}^d, t_{ijk}^d\}_{d \text{ s.t. } C_d \in \mathcal{I}},\ M_{ijk}\right)$$

to simulate the view of the corrupted parties in the execution for computing M_{ijk}.

Case 3: If $i = j = k$, only one server S_i is involved in computing M_{iii} and is uncorrupted. Thus the only message in the view of the corrupted parties is the message from S_i to O sending M_{iii}. Sim^m emulates the message using $M_{iii} \in \mathcal{M}$ computed or sampled in step Sampling III. Denote by Sim^1 this simulation procedure.

3. Finally, Sim^m simulates the execution of Π_{Add} by invoking the simulator $\mathsf{Sim}_{\mathsf{Add}}$ for Π_{Add}

$$\mathsf{Sim}_{\mathsf{Add}}\left(\mathcal{I},\ \left\{T_d = z_d + \sum_{ijk} t_{ijk}^d\right\}_{d \text{ s.t. } C_d \in \mathcal{I}},\ \left(y - \sum_{ijk} M_{ijk}\right)\right)\ .$$

If O is uncorrupted, Sim^m proceeds as described above, except for the following:

- In Step 1, Sim^m does not execute Sampling III.
- In Step 2, to simulate the executions computing M_{ijk}, in Case 1 and 2, Sim^m invokes Sim^3 and Sim^2 without M_{ijk}, and in Case 3, since O is uncorrupted, there are no messages to simulate.
- In Step 3, to simulate the execution of Π_{Add}, Sim^m invokes $\mathsf{Sim}_{\mathsf{Add}}$ without the output $y - \sum_{ijk} M_{ijk}$.

Correctness of Simulation: To show that the view generated by Sim^m is indistinguishable to the view of the corrupted parties in the real execution, we introduce the following intermediate hybrid simulation procedure:

$\mathsf{HSim}(\mathcal{I}, \{(x_d, z_d)\}_{d \in [3]}, y)$ with inputs of all clients proceeds as follows:

1. For every client C^d (honest and corrupted), HSim emulates perfectly its shares and blinding bits at random, that is, sampling $\{s_i^d\}$ at random subject to that $\sum_{i\in[m]} s_i^d = x_d$, and samples t_{ijk}^d at random for every $i,j,k \in [m]$. Then, it computes all M_{ijk} honestly

$$M_{ijk} = s_i^1 s_j^2 s_k^3 + t_{ijk}^1 + t_{ijk}^2 + t_{ijk}^3 \ .$$

2. For every $i,j,k \in [m]$, HSim simulates the view of the corrupted parties in the execution for computing M_{ijk} as Sim^m does, but using the above honestly generated $\{s_i^d, t_{ijk}^d\}$ as inputs and $\{M_{ijk}\}$ as outputs for $\mathsf{Sim}^3, \mathsf{Sim}^2, \mathsf{Sim}^1$.

3. HSim simulates the view of the corrupted parties in the execution of Π_{Add} as Sim^m does, using the above honestly generated $\{t_{ijk}^d\}$ as inputs and $y - \sum_{ijk} M_{ijk}$ as output for $\mathsf{Sim}_{\mathsf{Add}}$.

We first argue that the view of the corrupted parties in the real execution is indistinguishable to that generated by the hybrid simulator, that is,

$$\left\{\mathsf{Real}_{\Pi^m}(\mathcal{I},\ \{(x_d, z_d)\}_{d\in[3]})\right\}_\lambda \approx \left\{\mathsf{HSim}(\mathcal{I},\ \{(x_d, z_d)\}_{d\in[3]})\right\}_\lambda$$

The only difference between the hybrid simulation and the real execution is that in the former all executions of $\Pi^3, \Pi^2, \Pi^1, \Pi_{\mathsf{Add}}$ are simulated, whereas in the latter, they are executed honestly. Since HSim generates the inputs and outputs fed to $\mathsf{Sim}^3, \mathsf{Sim}^2, \mathsf{Sim}^1, \mathsf{Sim}_{\mathsf{Add}}$ honestly. It follows directly from the security of $\Pi^3, \Pi^2, \Pi^1, \Pi_{\mathsf{Add}}$ that the above indistinguishability holds.

Next, we argue that the hybrid simulation is indistinguishable to the actual simulation, that is,

$$\left\{\mathsf{HSim}(\mathcal{I},\ \{(x_d, z_d)\}_{d\in[3]})\right\}_\lambda = \{\mathsf{Sim}^m(\mathcal{I},\ \{(x_d, z_d)\}_{d \text{ s.t. } C_d\in\mathcal{I}},\ y)\}_\lambda$$

The only difference in the two simulation procedures lies in how the inputs and outputs fed to the simulators $\mathsf{Sim}^3, \mathsf{Sim}^2, \mathsf{Sim}^1, \mathsf{Sim}_{\mathsf{Add}}$ are generated. In HSim, they are generated honestly, whereas Sim^m generates them with the following difference:

- For every honest client C_d, and every corrupted server S_i, the share s_i^d is generated at random, not subject to being consistent with x_d.
- If not all clients are corrupted, for every i,j,k s.t. not all of S_i, S_j, S_k are corrupted, M_{ijk} is sampled at random instead of computed honestly.

We argue that the above sampling procedure produces the same distribution as in HSim. First, for every uncorrupted client C_d, the shares s_i^d corresponding to uncorrupted severs $S_i \notin \mathcal{I}$ are never used by neither Sim^m nor HSim. Therefore, the marginal distribution of s_i^d corresponding to corrupted severs $S_i \in \mathcal{I}$ are random. Second, for every i,j,k s.t. not all of S_i, S_j, S_k are corrupted, the blinding bits t_{ijk}^d belonging to the honest clients $C_d \notin \mathcal{I}$ are never used by neither Sim^m nor HSim. Therefore, the distribution of M_{ijk} is also random. Since the rest of the simulation is identical in HSim and Sim^m, we have that their distributions are identical.

By a hybrid lemma, we conclude the correctness of the simulator, which further concludes Lemma 5.7.

5.1.2 General Client-Server MPC for Any Functionality – Proof of Lemma 5.8

For any polynomials n, m, and any polynomial-time computable function $F : (\{0,1\}^*)^n \to \{0,1\}$, we construct a (n,m)-MPC protocol Π_F that securely computes F. To this end, we rely on the following tools:

- A *degree*-3 randomized encoding $(\mathsf{RE}_F, \mathsf{RE.Eval})$ for F (see Definition 5.10 below).
- A $(3,m)$-MPC protocol $\Pi_{\mathsf{3Mult\text{-}Plus}}$ that securely computes $\mathsf{3Mult\text{-}Plus}$, as guaranteed to exist by Lemma 5.7.
- A (n,m)-MPC protocol Π_{Add} that securely computes Add, as given by Claim 5.9.

Definition 5.10. A *randomized encoding scheme* $(\mathsf{RE}_F, \mathsf{RE.Eval})$ for a function $F : \{0,1\}^* \to \{0,1\}$ consists of two algorithms that behave as follows.

$\widehat{F(x)} \leftarrow \mathsf{RE}_F(x)$ probabilistically samples a string, which acts as a randomized encoding for the computation (F,x), in time polynomial in $T_{|x|}$, the worst-case running time of (a given implementation of) F on inputs of length $|x|$.

$y \leftarrow \mathsf{RE.Eval}(\widehat{F(x)})$, which is meant to deterministically compute a string y from the randomized encoding $\widehat{F(x)}$, in time polynomial in $|\widehat{F(x)}|$.

They satisfy the following conditions.

Correctness: For any $x \in \{0,1\}^*$,

$$\Pr[\ \mathsf{RE.Eval}(\mathsf{RE}_F(x)) \neq F(x)\ ] \leq \mathsf{negl}(|x|)\ ,$$

where the probability is taken over the randomness of RE_F.

Simulation: There exists an algorithm $\mathsf{RE.Sim}$ such that the following ensembles are computationally indistinguishable,[3]

$$\left\{\ \mathsf{RE}_F(x)\ \right\}_x \qquad \approx \qquad \left\{\ \mathsf{RE.Sim}(1^{|x|}, F(x))\ \right\}_x\ ,$$

where $\mathsf{RE.Sim}$ runs in time polynomial in $T_{|x|}$ as defined above.

The Protocol Π_F: Let $(\mathsf{RE}_F, \mathsf{RE.Eval})$ be the degree-3 randomized encoding scheme for the function F. To compute $F(x_1, \cdots, x_n)$, the n clients $C_1, \cdots, C_n$, with respective inputs $x_1, \cdots, x_n$, work with the m servers $S_1, \cdots S_m$ to enable the output client O to obtain a randomized encoding of the computation (F,x),

$$\widehat{F(x)} = \mathsf{RE}_F(x_1, \cdots, x_n\ ;\ r = r_1 + \cdots + r_n)\ ,$$

where the random tape r is the (bit-wise) sum of n random tapes $r_1, \cdots, r_n$ contributed by each client, and O can then evaluate the randomized encoding $\widehat{F(x)}$ to obtain the actual output $y = F(x)$.

[3]Here, the ensemble short-hand means that there exists a negligible function ν such that for every string $x \in \{0,1\}^*$, no attacker can tell apart $\mathsf{RE}_F(x)$ and $\mathsf{RE.Sim}(1^{|x|}, F(x))$ with advantage larger than $\nu(|x|)$.

Since RE_F is a degree 3 polynomial in the input bits (the x-bits) and the random bits (the r-bits), it can be expanded as sums of degree-3 monomials. Formally, for every bit $\ell \in [|\widehat{F(x)}|]$ of the randomized encoding,

$$\widehat{F(x)}_\ell = \sum_{v_i,v_j,v_k \in V} a^\ell_{ijk}\ v_i v_j v_k\ , \qquad \text{where } V = \left\{\ (x_{d,i})_{d\in[n],i\in[|x_1|]}\ \right\} \cup \left\{\ (r_{d,j})_{d\in[n],i\in[|r|]}\ \right\}\ ,$$

where a^ℓ_{ijk} is the coefficient associated with the degree 3 monomial $v_iv_jv_k$, and V is the set of all x-bits and r-bits of all clients. Thanks to the degree 3 structure, this sum can be computed using the protocols $\Pi_{\mathsf{3Mult\text{-}Plus}}$ and Π_{Add} as follows:

1. Each client C_d samples $r_d \leftarrow \{0,1\}^{|r|}$ at random, as its share of the random tape for computing the randomized encoding.

2. For every $\ell \in [|\widehat{F(x)}|]$, every $v_i, v_j, v_k \in V$, let $C_{d_i}, C_{d_j}C_{d_k}$ be the clients that *own* these bits respectively (i.e., if v_x is an input bit, C_{d_x} is the client that has this input bit; if v_x is a random bit, C_{d_x} is the client that sampled it). They compute the degree three monomial $a^\ell_{ijk}v_iv_jv_k$ blinded with random bits $t^{\ell,1}_{ijk}$, $t^{\ell,2}_{ijk}$ and $t^{\ell,3}_{ijk}$,

   $$M^\ell_{ijk} = \mathsf{3Mult\text{-}Plus}((a^\ell_{ijk}v_i, t^{\ell,1}_{ijk}), (v_j, t^{\ell,2}_{ijk}), (v_k, t^{\ell,3}_{ijk})) = a^\ell_{ijk}v_iv_jv_k + t^{\ell,1}_{ijk} + t^{\ell,2}_{ijk} + t^{\ell,3}_{ijk}\ .$$

   in the following two cases:

   Case 1: All $C_{d_i}, C_{d_j}C_{d_k}$ are distinct. In this case, they sample random blinding bits $t^{\ell,1}_{ijk}$, $t^{\ell,2}_{ijk}$, $t^{\ell,3}_{ijk}$ respectively, and execute the protocol $\Pi_{\mathsf{3Mult\text{-}Plus}}$ with all m servers, using inputs $(a^\ell_{ijk}v_i, t^{\ell,1}_{ijk})$, $(v_j, t^{\ell,2}_{ijk})$, and $(v_k, t^{\ell,3}_{ijk})$ respectively.

   Case 2: Not all $C_{d_i}, C_{d_j}C_{d_k}$ are distinct. In this case, each client C_d uses the product of its bits as its actual v bit. In addition, add other clients (chosen in an arbitrary way) to the computation, so that, there are exactly three clients; the added clients set their v bits to 1. Let $C_{d'_i}, C_{d'_j}C_{d'_k}$ be the three (distinct) clients. They proceed identically as in Case 1 to compute the term M^ℓ_{ijk}.

   At the end of this execution, O obtains term M^ℓ_{ijk}.

3. For every $\ell \in [|\widehat{F(x)}|]$, each client sums up the blinding t^ℓ-bits that it has sampled (related to the ℓ'th bit of the randomized encoding); denote the sum as T^ℓ_d. They, $C_1, \cdots, C_n$, run the protocol Π_{Add} with all servers $S_1, \cdots, S_m$ and O to compute their sum $T^\ell = \sum_{d\in[n]} T^\ell_d$. This execution is carried out in parallel with all executions in the previous step.

4. Finally, the output client O computes for every $\ell \in [|\widehat{F(x)}|]$, $\widehat{F(x)}_\ell = (\sum_{v_i,v_j,v_k\in V} M^\ell_{i,j,k} - T^\ell)$. Then, it decodes the randomized encoding $y = \mathsf{RE.Eval}(\widehat{F(x)})$ and outputs y.

Correctness: It is easy to verify that O indeed obtains the correct randomized encoding.

$$\begin{aligned}
\widehat{F(x)}_\ell &= \sum_{v_i,v_j,v_k\in V} \left(a^\ell_{ijk}v_iv_jv_k + t^{\ell,1}_{ijk} + t^{\ell,2}_{ijk} + t^{\ell,3}_{ijk}\right) - \sum_{v_i,v_j,v_k\in V} \left(t^{\ell,1}_{ijk} + t^{\ell,2}_{ijk} + t^{\ell,3}_{ijk}\right) \\
&= \sum_{v_i,v_j,v_k\in V} a^\ell_{ijk}v_iv_jv_k
\end{aligned}$$

Then, by the correctness of randomized encoding, the output $y = \mathsf{RE.Eval}(\widehat{F(x)})$ equals to $F(x)$.

The Simulator Sim_F of Π_F: To show the security of Π_F, we need to construct a simulator Sim_F, such that, for every possible corruption set $\mathcal{I}$, excluding at least one server, Sim_F can simulate the view of the corrupted players using only the inputs of the corrupted clients, and the output if the output client is corrupted. Formally,

$$\left\{\mathsf{Real}_{\Pi_F}(\mathcal{I},\ \{x_d\}_{d\in[n]})\right\}_\lambda \approx \left\{\mathsf{Sim}_F(\mathcal{I},\ \{x_d\}_{d \text{ s.t. } C_d\in\mathcal{I}},\ y')\right\}_\lambda, \qquad y' = \begin{cases} y = F(x_1,\cdots,x_n) & O\in\mathcal{I} \\ \perp & O\notin\mathcal{I} \end{cases}$$

We now formally describe the simulation procedure $\mathsf{Sim}_F(\mathcal{I}, \{(x_d)\}_{d \text{ s.t. } C_d\in\mathcal{I}}, y')$.

If O is corrupted, Sim_F proceeds as follows:

1. Sim_F samples the following:

   Sampling I: For every corrupted client $C_d \in \mathcal{I}$, Sim_F samples r_d at random, which is C_d's share of the random tape for computing the randomized encoding. In addition, for every ℓ, Sim_F samples all the blinding t^ℓ-bits generated by C_d, and sums them up to T_d^ℓ. (Note that after this step, Sim_F has all the input, random, and blinding bits of all corrupted clients.)

   Sampling II: For every $\ell \in [|\widehat{F(x)}|]$, and every $v_i, v_j, v_k \in V$, if all clients $C_{d'_i}, C_{d'_j}, C_{d'_k}$ involved for computing M_{ijk}^ℓ are corrupted, Sim_F compute honestly

   $$M_{ijk}^\ell = a_{ijk}^\ell v_i v_j v_k + t_{ijk}^{\ell,1} + t_{ijk}^{\ell,2} + t_{ijk}^{\ell,3}\ ,$$

   using the input, random, and blinding bits of all corrupted parties. Otherwise, if not all of $C_{d'_i}, C_{d'_j}, C_{d'_k}$ are corrupted, sample M_{ijk} at random.

2. For every $\ell \in [|\widehat{F(x)}|]$, and every $v_i, v_j, v_k \in V$, Sim_F simulates the execution for computing M_{ijk}^ℓ by invoking the simulators $\mathsf{Sim}_{\mathsf{3Mult\text{-}Plus}}$ for $\Pi_{\mathsf{3Mult\text{-}Plus}}$ as follows:

   Case 1: All $C_{d_i}, C_{d_j}C_{d_k}$ are distinct. In this case, let X_{ijk} be the subset of bits in $\{a_{ijk}^\ell v_i, v_j, v_k, t_{ijk}^{\ell,1}, t_{ijk}^{\ell,2}, t_{ijk}^{\ell,3}\}$ that are owned by corrupted clients $\{C_{d_i}, C_{d_j}C_{d_k}\} \cap \mathcal{I}$. It then invokes

   $$\mathsf{Sim}_{\mathsf{3Mult\text{-}Plus}}\left(\{C_{d_i}, C_{d_j}C_{d_k}\}\cap\mathcal{I},\ X_{ijk},\ M_{ijk}\right)$$

   to simulate the view of the corrupted parties in the execution for computing M_{ijk}.

   Case 2: Not all $C_{d_i}, C_{d_j}C_{d_k}$ are distinct. In this case, the protocol specifies three (distinct) clients $C_{d'_i}, C_{d'_j}, C_{d'_k}$ and their v bits for computing M_{ijk}^ℓ. Sim_F proceeds identically as above to simulate the view of the corrupted parties $\{C_{d'_i}, C_{d'_j}, C_{d'_k}\} \cap \mathcal{I}$.

3. Sim_F simulates the randomized encoding using the output $y' = y$, $\widetilde{F(x)} \leftarrow \mathsf{RE.Sim}(1^{|x|}, y)$.

4. Finally, for every $\ell \in [|\widehat{F(x)}|]$, Sim_F simulates the execution of Π_{Add} for computing the ℓ'th bit of the randomized encoding, by invoking the simulator $\mathsf{Sim}_{\mathsf{Add}}$ for Π_{Add},

   $$\mathsf{Sim}_{\mathsf{Add}}\left(\mathcal{I},\ \left\{T_d^\ell\right\}_{d \text{ s.t. } C_d\in\mathcal{I}},\ \sum_{v_i,v_j,v_k\in V} M_{ijk}^\ell - \widetilde{F(x)}_\ell\right)\ .$$

If O is uncorrupted, Sim_F proceeds as described above, except for the following:

- In Step 1, Sim_F does not execute the step Sampling II.
- In Step 2, to simulate the execution computing M^ℓ_{ijk}, Sim_F invokes $\mathsf{Sim}_{\mathsf{3Mult\text{-}Plus}}$ without M^ℓ_{ijk}.
- In Step 3, Sim_F skips this step.
- In Step 4, to simulate the executions of Π_{Add}, Sim_F invokes $\mathsf{Sim}_{\mathsf{Add}}$ without the output $\sum_{v_i,v_j,v_k \in V} M^\ell_{ijk} - \widetilde{F(x)}_\ell$.

Correctness of Simulation: To show that the view generated by Sim_F is indistinguishable to the view of the corrupted parties in the real execution, we introduce the following intermediate hybrid simulation procedure:

$\mathsf{HSim}(\mathcal{I}, \{(x_d)\}_{d\in[n]}, y)$ with inputs of all clients proceeds as follows:

1. For every client C^d (honest and corrupted), HSim emulates perfectly its random and blinding bits, that is, sample r_d and all t^ℓ-bits generated by C_d at random. Then, it computes all $M^\ell_{ijk} = a^\ell_{ijk} v_i v_j v_k + t^{\ell,1}_{ijk} + t^{\ell,2}_{ijk} + t^{\ell,3}_{ijk}$ honestly.
2. For every ℓ, and $v_i, v_j, v_k \in V$, HSim simulates the view of the corrupted parties in the execution for computing M^ℓ_{ijk} as Sim_F does (using the x-, r-, and t^ℓ- bits of the corrupted clients as inputs), except that it feeds $\mathsf{Sim}_{\mathsf{3Mult\text{-}Plus}}$ above honestly generated M^ℓ_{ijk} as output.
3. HSim computes honestly the randomized encoding $\widehat{F(x)} = \mathsf{RE}_F(x_1, \cdots, x_n\, ; r = r_1 + \cdots + r_n)$.
4. For every ℓ, HSim simulates the execution of Π_{Add} for computing the ℓ'th bit of the randomized encoding as Sim_F does, except that it feeds $\mathsf{Sim}_{\mathsf{Add}}$ the value $(\sum_{v_i,v_j,v_k} M^\ell_{ijk} - \widehat{F(x)}_\ell)$ as output.

We first argue that the view of the corrupted parties in the real execution is indistinguishable to that generated by the hybrid simulator, that is,

$$\left\{\mathsf{Real}_{\Pi_F}(\mathcal{I},\ \{(x_d,)\}_{d\in[n]})\right\}_\lambda \approx \left\{\mathsf{HSim}(\mathcal{I},\ \{(x_d)\}_{d\in[n]})\right\}_\lambda$$

The only difference between the hybrid simulation and the real execution is that in the latter all executions of $\Pi_{\mathsf{3Mult\text{-}Plus}}$ and Π_{Add} are simulated, whereas in the former, they are executed honestly. Observe that for every execution of $\Pi_{\mathsf{3Mult\text{-}Plus}}$ or Π_{Add}, the input and output that HSim feeds to $\mathsf{Sim}_{\mathsf{3Mult\text{-}Plus}}$ or $\mathsf{Sim}_{\mathsf{Add}}$ is identically distributed as the input and output of the execution in the real world. Then, indistinguishability follows directly from the security of $\Pi_{\mathsf{3Mult\text{-}Plus}}$ and Π_{Add}.

Next, we argue that the hybrid simulation is indistinguishable to the actual simulation, that is,

$$\left\{\mathsf{HSim}(\mathcal{I},\ \{(x_d)\}_{d\in[n]})\right\}_\lambda = \{\mathsf{Sim}_F(\mathcal{I},\ \{(x_d)\}_{d \text{ s.t. } C_d\in\mathcal{I}},\ y)\}_\lambda$$

The only difference in the two simulation lies in how the *outputs* fed to the simulators $\mathsf{Sim}_{\mathsf{3Mult\text{-}Plus}}$, and $\mathsf{Sim}_{\mathsf{Add}}$ are generated (the inputs fed to the simulators are identically distributed in these two simulation). In HSim, the outputs are computed honestly as in the real execution, whereas in Sim_F, they are simulated with the following difference:

- For every ℓ, and $v_i, v_j, v_k \in V$, such that, not all of the clients $C_{d'_1}, C_{d'_2}, C_{d'_3}$ involved in computing M^{ℓ}_{ijk} is corrupted, the output M^{ℓ}_{ijk} is sampled at random.

- For every ℓ, the output $(\sum_{v_i,v_j,v_k} M^{\ell}_{ijk} - \widetilde{F(x)}_{\ell})$ of the ℓ'th execution of Π_{Add} is generated using a simulated randomized encoding, instead of an honestly generated one $\widehat{F(x)}$.

We argue that the distribution of outputs described above is computationally indistinguishable to that used in HSim. First, for every M^{ℓ}_{ijk} computed by clients $C_{d'_1}, C_{d'_2}, C_{d'_3}$ that are not all corrupted, at least one of the blinding bits $\{t^{\ell,1}_{ijk}, t^{\ell,2}_{ijk}, t^{\ell,3}_{ijk}\}$ is never used in Sim_F and HSim. Thus, the marginal distribution of M^{ℓ}_{ijk} is random in both Sim_F and HSim. Second, by the security of the randomized encoding, the simulated randomized encoding used in Sim_F is computationally indistinguishable to the honestly generated one used in HSim.

Therefore, by a hybrid lemma, we conclude the correctness of the simulator, which further concludes Lemma 5.8.

5.1.3 Comparison with Other 2-Round MPC Constructions

The round complexity of multi-party computation protocols has been extensively studied. So far, in the optimal 2-round setting, we have 2-round protocols *in the Common Reference String (CRS) model*, based on LWE [AJLA$^+$12, MW16, CM15, BP16, PS16], and protocols *in the plain model,* from indistinguishability obfuscation or witness encryption, together with NIZK [GGHR14, GP15, CGP15, DKR15, GLS15], or bilinear groups [GS17a], or even 2-round semi-honest Oblivious Transfer (OT) protocols [GS17b, BL17].

Our construction above starts with a $(3, 2)$-HSS for computing $\mathsf{3Mult\text{-}Plus}$. Alternatively, we can start with a $(3, 3)$-MPC for computing $\mathsf{3Mult\text{-}Plus}$ and apply Lemmas 5.7 and 5.8 only. In the latter case, we extend the 2-round client-server MPC protocols in the Public Key Infrastructure (PKI) model based on DDH by [BGI17], from handling only a *constant* number of servers, to handling an *arbitrary* number of servers. Below, we briefly compare our construction with the above mentioned 2-round protocols.

- *Approach and Underlying assumptions:* The LWE-based protocols use multi-key FHE. By using multi-input HSS instead, our construction can be based on DDH. The constructions in the plain model use a completely different approach: At a very high-level, they obtain 2-round protocols by collapsing rounds of multi-round protocols. The state-of-the-art constructions in this line [GS17b, BL17] managed to perform round-collapsing using "distributed garbling", which can be done relying only on the necessary assumption of 2-round semi-honest OT. Though our protocols rely on stronger assumptions, the HSS-based approach has other advantages as discussed below.

- *Client-server model:* Our protocols are in the more general client-server model. Note that any (n, n)-MPC with the same number of clients and servers implies n-party MPC in the standard model, by letting each party act as one client, one server, and one output client. However, the converse is not true in general. In a 2-round n-party MPC protocol, i) the first round messages may depend on the function being computed, ii) the second-round messages may depend on the private inputs of the parties, and their random coins for generating the first-round messages, and iii) recovering the outputs may require private inputs and coins.

In particular, condition ii) is true for all known protocols following the round-collapsing approach. In contrast, such dependency is not allowed in the client-server model. Therefore, client-server MPC provides more flexibility. For instance, the clients can share their inputs "off-line" before the function is chosen, and servers upon learning the functions can perform the computation homomorphically without learning any information of clients' private inputs, and finally any party who has collected the output shares from the servers can recover the output.

- *Asymptotic Efficiency:* Known protocols following the round-collapsing approach have high asymptotic complexity: The overall computational complexity (of all parties) scales linearly in the complexity T of the functionality being computed, but scales with n^4 in the number of parties. In two natural settings described below, the complexity of our protocols scales with n^3 only.
  - The first setting is when the number of servers is a constant $m = O(1)$. In this case, starting from a $(3, m)$-MPC protocol $\Pi_{\mathsf{3Mult\text{-}Plus}}$ for $\mathsf{3Mult\text{-}Plus}$ for a constant number of servers by [BGI17], we can obtain a (n, m)-MPC protocol Π_F for an arbitrary functionality F by only applying Lemma 5.8 (skipping the step of applying Lemma 5.7 for increasing the number of servers to super-constant). Recall that in Lemma 5.8, we reduce the task of computing F to the task of computing a degree-3 randomized encoding of F using random coins contributed by all clients,

$$\widehat{F(x)} = \mathsf{RE}_F(x_1, \cdots, x_n \ ; \ r = r_1 + \cdots + r_n) = \sum a^{\ell}_{ijk} \ v_i v_j v_k \ , \tag{2}$$

    where v_i is either a bit in some input x_l or some random tape r_l. Since each degree-3 v-monomial can be computed via an invocation of $\Pi_{\mathsf{3Mult\text{-}Plus}}$ (and each invocation of $\Pi_{\mathsf{3Mult\text{-}Plus}}$ has fixed polynomial complexity $\mathsf{poly}(\lambda)$), the complexity of our protocols is determined by the number of such degree-3 monomials. Since RE_F has complexity $T\mathsf{poly}(\lambda)$, there are at most $T\mathsf{poly}(\lambda) \times n^3$ such monomials.
  - The second setting is when the number of servers is the same as the number of clients $n = m$ and the indexes of the set of corrupted servers is the same as that of corrupted clients. This setting in particular implies standard n-party MPC. In this setting, starting from a $(3, 3)$-MPC protocol $\Pi_{\mathsf{3Mult\text{-}Plus}}$ for $\mathsf{3Mult\text{-}Plus}$ (and a $(2, 2)$-MPC protocol for $\mathsf{3Mult\text{-}Plus}$), we can again obtain a (n, m)-MPC protocol Π_F for F *without* applying Lemma 5.7. To do so, we modify Lemma 5.8 as follows. Again, we use $\Pi_{\mathsf{3Mult\text{-}Plus}}$ to compute every v-monomial in the randomized encoding in Equation (2). However, for each v-monomial, the three clients C_i, C_j, C_k holding v_i, v_j, v_k compute the monomial with servers S_i, S_j, S_k with the *same* indexes (instead of with all servers). (In case that only 2 of i, j, k are distinct, invoke a $(2, 2)$-MPC for $\mathsf{3Mult\text{-}Plus}$, and in case that they are all the same, simply send the computed output to the server.) Since the indexes of corrupted servers are the same as that of corrupted clients, whenever clients C_i, C_j, C_k are not all corrupted, the $\mathsf{3Mult\text{-}Plus}$ computation is secure, and hence security holds. With this modification, the overall complexity is again determined by the number of v-monomials, which scales with $T\mathsf{poly}(\lambda)n^3$.

Finally, we remark that if there exist degree-3 randomized encodings, where every monomial has only degree 2 in the random coins, the number of v-monomials to be computed decreases

to $T\mathsf{poly}(\lambda)n^2$. This improves the asymptotic efficiency of our protocols, but has no effect on the efficiency of these round-collapsing protocols. However, currently, the existence of such randomized encodings is unknown.

- *Output Client Complexity:* Finally, we remark that our protocols has the feature that the output client is relatively efficient. Its sole job is recovering the randomized encoding from the output shares and then decode the randomized encoding. The latter task has $T\mathsf{poly}(\lambda)$ complexity, and the former has at most $T\mathsf{poly}(\lambda) \times m$ complexity, since for every output element, the output client merely needs to add the corresponding output shares from the m servers, due to the additive decoding of HSS. This feature is important for delegating secure computation. Computationally weak (input) clients can share their inputs offline, and computationally weak output clients can recover the outputs efficiently. The most expensive computation is performed by the servers who are computationally powerful. In comparison, protocols following the round-collapsing approach all have high complexity for deriving the outputs, namely $T\mathsf{poly}(\lambda) \times n^3$ per party.

5.2 Worst-Case to Average-Case Reductions

In this section we describe a simple application of HSS to worst-case to average-case reductions. We then discuss applications of these reductions to fine-grained average-case hardness and verifiable computation. These applications of HSS can be seen as more efficient or more general conditional variants of previous applications of locally random reductions that rely on arithmetization or error-correcting codes [Lip89, BK89, BF90, BFNW93, STV01, GI14, BRSV17]. In contrast to the above reductions, the HSS-based reductions can reduce any polynomial-time computable function to another polynomial-time computable function with closely related complexity.

Worst-case to average-case reductions based on fully homomorphic encryption (FHE) were previously used by Chung et al. [CKV10] in the context of delegating computations. Compared to the FHE-based reductions, the use of HSS has the advantages of diversifying assumptions, making only a constant number of queries to a *Boolean* function (as small as 2), and minimizing the complexity of recovering the output from the answers to the queries. The latter can lead to efficiency advantages in the context of applications.

To make the discussion concrete, we focus here on the application of (computationally secure, additive[4]) $(1, 2)$-HSS for circuits, namely for the universal function $F(C; x) = C(x)$. Such HSS schemes can be based on variants of the LWE assumption (see Appendix B). Weaker versions of the following results that apply to branching programs can be based on the DDH assumption or the circular security of Paillier encryption using the HSS schemes from [BGI16a, FGJS17].

A high level overview. The idea of using HSS for worst-case to average-case reductions is similar to previous applications of locally random reductions for this purpose, except that we apply a "hybrid HSS" technique [BGI16a] to improve the efficiency of the reduction. Concretely, the reduction proceeds as follows. Suppose for simplicity that the HSS sharing algorithm $\mathsf{Share}(1^\lambda, x)$ outputs a pair of shares (x^1, x^2) such that each share is individually pseudo-random. Moreover, suppose that the evaluation function $\mathsf{Eval}(j, C, x^j)$ does not depend on j. The evaluation of a circuit $C : \{0,1\}^n \to \{0,1\}$ on an arbitrary input $x \in \{0,1\}^n$ can then be reduced to the evaluation of an extended circuit $\hat{C}$, defined by $\hat{C}(\hat{x}) = \mathsf{Eval}(C, \hat{x})$, on the two inputs x^1, x^2. Indeed, $C(x) =$

[4]The requirement of being additive can be relaxed here to small decoding complexity.

$\hat{C}(x^1) \oplus \hat{C}(x^2)$. Now, suppose that $\hat{C}^*$ is a polynomial-size circuit that agrees with $\hat{C}$ on all but an ϵ fraction of the inputs. Then, by the pseudo-randomness of x^1, x^2, the probability that $\hat{C}^*$ agrees with $\hat{C}$ on both inputs, and hence the reduction outputs the correct value $C(x)$, is at least $1 - 2\epsilon - \mathsf{negl}(n)$. Finally, to make the reduction run in near-linear time, we convert the given HSS into a hybrid HSS scheme in which the sharing Share' can be implemented in near-linear time. The algorithm Share' uses Share to share a short seed r for a pseudorandom generator G, and includes the masked input $G(r) \oplus x$ as part of both shares. Given a circuit C and $G(r) \oplus x$, one can efficiently compute a circuit C' such that $C'(r) = C(x)$. The algorithm Eval' of the hybrid scheme applies Eval to homomorphically evaluate C' on r.

The following theorem formalizes and generalizes the above. Here, by a "near-linear time" algorithm we refer to an algorithm whose running time is $O(n^{1+\epsilon})$ for an arbitrary $\epsilon > 0$.

Theorem 5.11 (Worst-case to average-case reductions from HSS)**.** *Suppose there is a* $(1, 2)$*-HSS scheme* $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ *for circuits. Then, there is a near-linear time probabilistic oracle algorithm* $Q^{[\cdot]} : \{0,1\}^* \to \{0,1\}^*$*, polynomial-time algorithm* $A : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$*, and a PPT sampling algorithm* $D(1^n)$ *with the following properties:*

- Q *makes two queries to a Boolean oracle (where the queries are computed in near-linear time and the answers are 1-bit long) and outputs the exclusive-or of the two answer bits.*
- *For any* $x \in \{0,1\}^n$ *and circuit* $C : \{0,1\}^n \to \{0,1\}$*, we have* $\Pr[Q^{A(C,\cdot)}(x) = C(x)] = 1$.
- *For any polynomial* $p(\cdot)$ *there is a negligible* $\mu(\cdot)$ *such that the following holds. For any* $x \in \{0,1\}^n$ *and circuits* $C : \{0,1\}^n \to \{0,1\}$*,* $A^*_C : \{0,1\}^n \to \{0,1\}$ *of size* $\leq p(n)$ *such that* $\Pr_{\hat{x} \leftarrow D(1^n)}[A^*_C(\hat{x}) = A(C, \hat{x})] \geq 1 - \epsilon$*, we have* $\Pr[Q^{A^*_C(\cdot)}(x) = C(x)] \geq 1 - 2\epsilon - \mu(n)$.

Moreover, if Share *produces pseudorandom shares then the distribution* $D(1^n)$ *can be replaced by the uniform distribution.*

Proof. In the following we treat the security parameter λ as being independent of the input length $n = |x|$, under the understanding that the security of the HSS is maintained whenever n is polynomially bounded in λ. This means that complexity of $n \cdot \mathsf{poly}(\lambda)$ qualifies as near-linear since we can set $\lambda = n^\delta$ for an arbitrarily small constant $\delta > 0$.

We start by converting $\Pi(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ into $\Pi'(\mathsf{Share}', \mathsf{Eval}', \mathsf{Dec})$ in which $\mathsf{Share}'(1^\lambda, x)$ runs in near-linear time. This is done using the following hybrid HSS approach (cf. [BGI16a]): $\mathsf{Share}'(1^\lambda, x)$ picks a random seed $r \in \{0,1\}^\lambda$ for a pseudorandom generator (PRG) G, lets $(r^1, r^2) \leftarrow \mathsf{Share}(1^\lambda, r)$, and outputs $((r^1, G(r) \oplus x), (r^2, G(r) \oplus x))$. (Note that the existence of G follows from the existence of a one-way function, which follows from additive HSS for circuits [GI14], and that $G(r)$ can be computed in time $n \cdot \mathsf{poly}(\lambda)$.) Given a circuit C and $G(r) \oplus x$, one can efficiently compute a circuit C' such that $C'(r) = C(x)$. The algorithm Eval' applies Eval to homomorphically evaluate C' on r.

We are now ready to define the algorithms Q, A, and D. Algorithm Q on input $x \in \{0,1\}^n$ lets $\lambda = n^\delta$ (for an arbitrarily small constant $\delta > 0$) and lets $(x^1, x^2) \leftarrow \mathsf{Share}'(1^\lambda, x)$. It then invokes the oracle twice, once on input $(1, x^1)$ and once on input $(2, x^2)$, and outputs the exclusive-or of the two answers. Algorithm A, on input $(C, (j, x^j))$, simply outputs $\mathsf{Eval}'(j, C, x^j)$. Finally, algorithm D, on input 1^n, lets $(x^1, x^2) \leftarrow \mathsf{Share}'(1^\lambda, 0^n)$ (for $\lambda = n^\delta$) and outputs (j, x^j) for a random $j \in \{1, 2\}$.

It is easy to see that Q, A, D described above satisfy the first two requirements (the second requirement follows from the correctness of Π' as an additive HSS scheme for circuits). We argue that the third requirement is also satisfied. The security of Π' implies that if $\Pr_{\hat{x}\leftarrow D(1^n)}[A^*_C(\hat{x}) = A(C,\hat{x})] \geq 1-\epsilon$ then, replacing $D(1^n)$ by $D'(x)$ that invokes $\mathsf{Share}'(1^\lambda, x)$ instead of $\mathsf{Share}'(1^\lambda, 0^n)$, we have $\Pr_{\hat{x}\leftarrow D'(x)}[A^*_C(\hat{x}) \neq A(C,\hat{x})] \leq \epsilon + \mu'(n)$ for a negligible μ'. Letting ϵ_j be the latter probability conditioned on $\hat{x} = (j, x^j)$ (namely, $\hat{x}$ being the share of server j) we have that $(\epsilon_1 + \epsilon_2)/2 \leq \epsilon + \mu'(n)$. Since the probability of Q failing is upper bounded by the probability that A^*_C differs from $A(C,\cdot)$ on at least one of the two queries, by a union bound we have $\Pr[Q^{A^*_C(\cdot)}(x) \neq C(x)] \leq \epsilon_1 + \epsilon_2 \leq 2\epsilon + 2\mu'(n)$ as required.

Finally, the "moreover" part of the theorem follows from the fact that when each of the shares x^1 and x^2 is individually pseudorandom, the distribution (j, x^j) for a random $j \in \{1,2\}$ is also pseudorandom, and hence the output of $D(1^n)$ is indistinguishable from a uniform. □

Remark 5.12 (Instantiating Theorem 5.11)**.** The strong flavor of HSS required by Theorem 5.11 can be instantiated under a variant of the LWE assumption that further assumes circular security [Gen09, DHRW16] (see Corollary B.3). Due to the negligible decoding error of the HSS, we get a slightly weaker version of the conclusion where $\Pr[Q^{A(C,\cdot)}(x) = C(x)] \geq 1 - \mathsf{negl}(n)$. On the other hand, since the implementation of Eval has a small asymptotic overhead, we get the stronger guarantee that the oracle $A(C,\cdot)$ has roughly the same circuit size as C (rather than being polynomially bigger). One can relax the assumption to a more standard variant of LWE by using *depth-dependent* HSS for circuits, where the length of the input shares grows polynomially with the depth of the circuit C being evaluated by Eval. In this case, using an LWE-based NC^1 implementation of the PRG G, the conclusion of Theorem 5.11 still holds when restricted to NC-circuits C. More generally, the complexity of Q should in this case be allowed to grow with the depth of C.

A straightforward generalization of Theorem 5.11 relaxes the assumption to $(1, m, 1)$-HSS, where the 2ϵ in the conclusion is changed to $m\epsilon$. However, this relaxation is not known to give rise to new HSS schemes for circuits.

We now informally discuss two types of applications of Theorem 5.11, which follow previous applications of such worst-case to average-case reductions from the literature.

Fine-grained average-case hardness. Theorem 5.11 implies, assuming HSS for circuits, that the following holds for any constants $c' > c$. For every polynomial-time computable function f there is a polynomial-time computable "extension" $\hat{f}$, such that if $\hat{f}$ has a time-$O(n^c)$ algorithm that computes it correctly on, say, 90% on the inputs, then f has a time-$O(n^{c'})$ probabilistic algorithm that computes it correctly (with overwhelming probability) on every input. This implies that if f is hard in the worst case for time $O(n^{c'})$ then $\hat{f}$ is hard in the average case for time $O(n^c)$. The same connection holds also in a non-uniform setting.

A similar result under the incomparable assumption that FHE exists is given in [CKV10]. These results are incomparable to recent results on fine-grained average case hardness [BRSV17, GR17] that obtain tighter and unconditional connections of this kind, but only for specific functions f. They are also incomparable to the unconditional average case time hierarchy from [GGH94], which does not apply to the probabilistic or nonuniform setting (see [GR17] for a more detailed comparison). We note that $\hat{f}$ is essentially the HSS Eval function applied to the circuits corresponding to f. Using LWE-based HSS (or FHE) constructions, the worst-case circuit complexity of $\hat{f}$ can be

made close to that of f (concretely, if f has circuits of size $O(n^d)$ then $\hat{f}$ has circuits of size $n^{d'}$ for any $d' > d$).

Verifiable computation. The goal of program checking [BK89] is to reliably compute a given function f using an untrusted program or piece of hardware that purportedly computes f. We consider a variant of the problem in which a program M for computing $f : \{0,1\}^n \to \{0,1\}$ can access a purported implementation of a related function $\hat{f}$. The program M can make oracle calls to $\hat{f}$ and perform additional computations, as long as the complexity of these additional computations is significantly smaller than that of computing f from scratch. The requirements are that if $\hat{f}$ is implemented correctly, then $M^{\hat{f}}(x) = f(x)$ for all x. On the other hand, even if $\hat{f}$ is replaced by an incorrect implementation $\hat{f}^*$, the output of $M^{\hat{f}^*}(x)$ on every input x is either $f(x)$ or $\perp$ except with small failure probability ϵ. This is very similar to the traditional goal of verifiable computation, except that a malicious "prover" $\hat{f}^*$ is required to be stateless. In this setting, one can make a direct use of probabilistically checkable proofs (PCPs) for proving the correctness of $f(x)$ without any additional cryptographic machinery.

Worst-case to average-case reductions provide a the following "amortized" approach. Suppose that computing $f(x)$ on the worst case reduces to computing a related function $\hat{f}(\hat{x})$ on the average case. Then, a checker M for f given a purported implementation $\hat{f}^*$ of $\hat{f}$ can proceed in the following way. In an offline phase, before any input x is known, M picks a random set of polynomially many inputs and checks that $\hat{f}^*$ correctly implements $\hat{f}$ on these inputs by evaluating $\hat{f}$ on its own. If any inconsistency is found M rejects. Otherwise, M is assured that (except with negligible probability) $\hat{f}^*$ correctly implements $\hat{f}$ on all but a $1/p(n)$ fraction of the inputs, for some polynomial $p(\cdot)$. This offline phase can now be used to verify any number of online instances: given an input x, the checker M applies the worst-case to average case reduction for computing $f(x)$ using oracle calls to $\hat{f}^*$. Since $\hat{f}^*$ is guaranteed to be correct on almost all inputs, the reduction will fail with small error probability, that can be exponentially reduced via repetition.

Combined with the worst-case to average-case reduction based on HSS from Theorem 5.11, this approach gives rise to checkers M with the following feature: after an input-independent polynomial-time preprocessing, any computation $f(x)$ can be verified with an arbitrarily small inverse polynomial error by receiving just a constant number of bits from $\hat{f}^*$. We do not know of any other approach for verifiable computation that yields such a result.

6 Conclusions and Open Problems

In this work we initiate a systematic study of homomorphic secret sharing (HSS) by providing a taxonomy of HSS variants and establishing some negative results and relations with other primitives. We also present applications of HSS in cryptography and complexity theory.

There is much left to understand about the feasibility and efficiency of HSS in different settings. In the information-theoretic setting, we have no strong negative results for *single-input,* (weakly) *compact* HSS. This should be contrasted with *multi-input* compact HSS, for which negative results are obtained in this work, and with single-input *additive* HSS, where information-theoretic impossibility results are also known [CGKS98]. The difficulty of making progress on this question can be partially explained by its relation with information-theoretic private information retrieval and locally decodable codes [KT00, BIKO12], for which proving good lower bounds is still an outstanding challenge. However, this barrier only seems to apply to special instances of the general problem. In

the computational setting, the main open problems are to obtain HSS schemes for circuits under new assumptions and, more broadly, extend the capabilities of HSS schemes that do not rely on FHE.

Acknowledgements. We thank the anonymous ITCS reviewers for helpful comments.

E. Boyle was supported by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069, and ERC grants 307952, 742754. N. Gilboa was supported by ISF grant 1638/15, a grant by the BGU Cyber Center by the European Union's Horizon 2020 ICT program (Mikelangelo project), and ERC grant 742754. Y. Ishai was supported by ERC grant 742754, NSF-BSF grant 2015782, BSF grant 2012366, ISF grant 1709/14, DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the DARPA through the ARL under Contract W911NF-15-C-0205. H. Lin was supported by NSF grants CNS-1528178, CNS-1514526, CNS-1652849 (CAREER), a Hellman Fellowship, the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No. W911NF-15-C-0236, and a subcontract No. 2017-002 through Galois. S. Tessaro was supported by NSF grants CNS-1553758 (CAREER), CNS-1423566, CNS-1719146, CNS-1528178, and IIS-1528041, and by an Alfred P. Sloan Research Fellowship. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *FOCS 2004*, pages 166–175, 2004.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. In *CCC*, pages 260–274, 2005.

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

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS*, pages 2105–2122, 2017.

[Ben86] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of A secret sharing. In *CRYPTO 1986*, pages 251–260, 1986.

[BF90] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *STACS*, pages 37–48, 1990.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[BGI15] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT*, pages 337–367, 2015.

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO*, pages 509–539, 2016. Full version: IACR Cryptology ePrint Archive 2016: 585 (2016).

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS 2016*, pages 1292–1303, 2016.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In *EUROCRYPT*, pages 163–193, 2017.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In *ITCS*, 2018.

[BGKL03] László Babai, Anna Gál, Peter G. Kimmel, and Satyanarayana V. Lokam. Communication complexity of simultaneous messages. *SIAM J. Comput.*, 33(1):137–166, 2003.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

[BIKO12] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Ilan Orlov. Share conversion and private information retrieval. In *CCC 2012*, pages 258–268, 2012.

[BIW10] Omer Barkol, Yuval Ishai, and Enav Weinreb. On d-multiplicative secret sharing. *J. Cryptology*, 23(4):580–593, 2010.

[BK89] Manuel Blum and Sampath Kannan. Designing programs that check their work. In *STOC 1989*, pages 86–97, 1989.

[BL17] Fabrice Benhamouda and Huijia Lin. k-round MPC from k-round OT via garbled interactive circuits. Manuscript, 2017.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *CRYPTO, Part I*, pages 190–213, 2016.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *STOC 2017*, pages 483–496, 2017.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, pages 143–202, 2000.

[CC06] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *CRYPTO*, pages 521–536, 2006.

[CCCX09] Ignacio Cascudo Pueyo, Hao Chen, Ronald Cramer, and Chaoping Xing. Asymptotically good ideal linear secret sharing with strong multiplication over *Any* fixed finite field. In *CRYPTO*, pages 466–486, 2009.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *STOC*, pages 11–19, 1988.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

[CG97] Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *STOC 1997*, pages 304–313, 1997.

[CGKS98] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CGP15] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In *TCC, Part II*, pages 557–585, 2015.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO 2010*, pages 483–501, 2010.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In *CRYPTO*, pages 630–656, 2015.

[Cou17] Geoffroy Couteau. Personal communication, 2017.

[DG15] Z. Dvir and S. Gopi. 2-server PIR with sub-polynomial communication. In *STOC*, pages 577–584, 2015.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *CRYPTO*, pages 93–122, 2016.

[DKR15] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In *TCC, Part II*, pages 586–613, 2015.

[Efr09] Klim Efremenko. 3-query locally decodable codes of subexponential length. In *STOC*, pages 39–44, 2009.

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith. Homomorphic secret sharing from paillier encryption. In *ProvSec*, pages 381–399, 2017.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710, 1992.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH94] Mikael Goldmann, Per Grape, and Johan Håstad. On average time hierarchies. *Inf. Process. Lett.*, 49(1):15–20, 1994.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.

[GI14] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *Proc. EUROCRYPT 14*, pages 640–658, 2014.

[GKST06] Oded Goldreich, Howard J. Karloff, Leonard J. Schulman, and Luca Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. *Computational Complexity*, 15(3):263–296, 2006.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO, Part II*, pages 63–82, 2015.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography — Basic Applications*. Cambridge University Press, 2004.

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In *TCC, Part II*, pages 614–637, 2015.

[GR17] Oded Goldreich and Guy N. Rothblum. Worst-case to average-case reductions for subclasses of p. *Electronic Colloquium on Computational Complexity (ECCC)*, 17-130, 2017.

[GS17a] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two round MPC from bilinear maps. In *FOCS 2017*, 2017.

[GS17b] Sanjam Garg and Akshayaram Srinivasan. Two-round secure multiparty computation from minimal assumptions. Manuscript, 2017.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO (1)*, pages 75–92, 2013.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[JRS17] Aayush Jain, Peter M. R. Rasmussen, and Amit Sahai. Threshold fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2017:257, 2017.

[KdW04] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *J. Comput. Syst. Sci.*, 69(3):395–420, 2004.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

[KNR99] Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *Proc. FOCS 97*, pages 364–373, 1997.

[KS92] Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM J. Discret. Math.*, 5(4), November 1992.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC*, pages 80–86, 2000.

[Lip89] Richard J. Lipton. New directions in testing. In *DIMACS Workshop on Distributed Computing And Cryptography*, pages 191–202, 1989.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, pages 735–763, 2016.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from lwe, revisited. In *TCC, Part II*, pages 217–238, 2016.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of secure computation*, pages 169–179. 1978.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[STV01] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proc. EUROCRYPT 2010*, pages 24–43, 2010.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[Yek07] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proc. STOC*, pages 266–274, 2007.

A Preliminaries

A.1 Learning With Errors Assumption

Below we identify $\mathbb{Z}_q$ with the symmetric interval $[-q/2, q/2) \bigcap \mathbb{Z}$ and let $[x]_q$ denote the reduction of x modulo q into this interval.

Definition A.1 (α-LWE [Reg09])**.** Let $n = n(\lambda), q = q(\lambda) \in \mathbb{Z}$ be functions of the security parameter λ and $\chi = \{\chi(\lambda)\}_\lambda$ be a distribution ensemble over $\mathbb{Z}$. The decision-LWE assumption with parameters (n, q, χ) states that for any polynomial $m = m(\lambda) \in \mathbb{Z}$, the following two distribution ensembles are computationally indistinguishable:

$$LWE[n, m, q, \chi] := \{(A, \vec{b}) : A \leftarrow \mathbb{Z}_q^{n\times m}, \vec{s} \leftarrow \mathbb{Z}_q^n, \vec{e} \leftarrow \chi^m, b = [\vec{s}A + \vec{e}]_q\},$$
$$U[n, m, q] := \{(A, \vec{b}) : A \leftarrow \mathbb{Z}_q^{n\times m}, \vec{b} \leftarrow \mathbb{Z}_q^m\} \text{ (i.e., uniform over } \mathbb{Z}_q^{(n+1)\times m}).$$

For $\alpha = \alpha(\lambda) \in (0, 1)$, the α-DLWE assumption asserts the existence of parameters n, q, χ as above with polynomial n in λ, such that $e \leftarrow \chi$ yields $|e| < \alpha q$ with overwhelming probability.

Note that the α-DLWE assumption becomes stronger as α gets smaller, but is commonly believed to hold for super-polynomially small α.

A.2 Spooky Encryption

"Spooky" encryption is a form of public-key encryption which admits certain types of limited malleability across independent keys (so-called "spooky action at a distance") [DHRW16]. This is denoted via an additional "spooky-eval" algorithm which takes as input k ciphertexts under independent keys and outputs k ciphertexts whose decryption under the respective keys satisfies a given relation of the original plaintext values. We present a treatment of *additive-function-sharing* spooky encryption as per [DHRW16], supporting relations of the form $\bigoplus_{i=1}^k y_i = C(x_1, \ldots, x_k)$ for circuit C of choice.

Definition A.2 (AFS-Spooky Encryption)**.** An *AFS-spooky encryption scheme* is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SpookyEval})$ such that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a semantically secure PKE scheme, and $\mathsf{SpookyEval}$ has syntax:

- $\mathsf{SpookyEval}(C, (\mathsf{pk}_i, c_i)_{i=1}^k) = (c'_1, \ldots, c'_k)$: Given a description of a circuit $C : (\{0,1\}^*)^k \to \{0,1\}^k$ with $k = k(\lambda)$ inputs and outputs, and k pairs of public keys and ciphertexts (pk_i, c_i), the procedure $\mathsf{SpookyEval}$ outputs k ciphertexts $c'_1, \ldots, c'_k$.

satisfying the following property:

- **AFS-spooky correctness:** There exists a negligible function ν such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C computing an k-argument function $f : (\{0,1\}^*)^k \to \{0,1\}$, and any set of inputs $x_1, \ldots, x_k$ for C, it holds that

$$\Pr\left[\begin{array}{ll} \forall i \in [k], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda), c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, x_i), & \\ (c'_1, \ldots, c'_k) \leftarrow \mathsf{SpookyEval}(C, (\mathsf{pk}_i, c_i)_{i=1}^k), & : \bigoplus_{i=1}^k y_i = C(x_1 \ldots, x_k) \\ \forall i \in [k], y_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, c'_i) & \end{array}\right] \geq 1 - \nu(\lambda).$$

We consider also a *leveled* variant, in which key generation Gen receives an additional depth parameter $d \in \mathbb{N}$ and spooky evaluation correctness holds only for circuits C of depth $\leq d$.

Further, an AFS-spooky scheme *in the common random string (CRS) model* has an additional algorithm CRSGen which takes the security parameter and outputs a random string crs of appropriate length. The algorithm Gen now also takes crs as input, and the AFS-spooky correctness and standard decryption properties hold over the choice of $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$.

Theorem A.3 ([DHRW16])**.** *Assuming the hardness of α-DLWE for any $\alpha(\lambda) \in \lambda^{-\omega(1)}$, there exists a leveled AFS-spooky encryption scheme in the CRS model. Further making a circular-security assumption, there exists a (non-leveled) AFS-spooky encryption scheme in the CRS model.*

A.3 Secure Multiparty Computation

We refer the reader to [Can00, Gol04] for standard definitions of MPC protocols. In a standard MPC protocol there are n parties who interact with each other in order to compute a function of their inputs. We say that such a protocol is *(semi-honest) secure* if it is computationally secure against a static, passive adversary who may corrupt any strict subset of the parties.

In this work, we consider the stronger notion of client-server MPC protocols, which is a special form of the standard MPC protocols where parties act in different roles, namely clients, servers, and output client, and communicate according to a special pattern.

Client-server MPC protocols. A client-server MPC protocol with n clients and m servers for computing $F : (\{0,1\}^*)^n \to \{0,1\}$ is a standard $(n+m+1)$-party MPC protocol with the following special form:

- n parties act as n clients $C_1, \cdots, C_n$, who each has an input x_i.
- m parties act as m servers $S_1, \cdots, S_m$, who do not have inputs (or equivalently, each has input $\perp$).
- One party acts as an output client O, who also does not have input.
- Clients, servers, and the output client interact with each other via secure point-to-point channels in order to compute $F(x_1, \cdots, x_n)$, observing the following canonical communication pattern:
  - In the first round each client sends a message, also called an *input share*, to each server.
  - Then there may be $r \geq 0$ rounds of interaction in which each server can send a message to each other server.
  - Finally, in the last, output reconstruction round, each server sends a message, also called an *output share*, to the output client, who computes an output by applying a local decoding function to the k messages it received.

In this work, we construct 2-round client-server MPC protocols Π for general functions $F : (\{0,1\}^*)^n \to \{0,1\}$; in such protocols, the servers do not interact with each other (i.e., $r = 0$), and the protocol involves only each client sending one input share to each server, followed by each server sending one output share to the output client.

Security. We say that a client-server MPC protocol Π is a *t-secure* protocol for computing F, if it is secure against a static, passive (semi-honest) adversary who may corrupt any set of parties

that includes at most t servers and an arbitrary number of clients. Security is defined identically as security of standard MPC. More specifically, there exists a simulator Sim, such that, for every set of corrupted parties $\mathcal{I} \subseteq \{C_1, \cdots, C_n, S_1, \cdots, S_m, O\}$ that include at most t servers, Sim can simulate the view of the corrupted parties using only the inputs of the corrupted clients, and the output if the output client is corrupted. Formally, let $\mathsf{Real}_\Pi(\mathcal{I}, \{(x_d)\}_{d\in[n]})$ denote the view of the corrupted parties $\mathcal{I}$ in an execution of Π with client inputs $\{x_d\}$. The simulator Sim must satisfy that for every sequence of corrupt sets $\{\mathcal{I}\}_{\lambda\in\mathbb{N}}$ and every sequence of tuples of inputs $\{\{x_d\}_{d\in[n]}\}_{\lambda\in\mathbb{N}}$ of length polynomial in λ, the following two ensembles are indistinguishable.

$$\left\{\mathsf{Real}_\Pi(\mathcal{I},\ \{x_d\}_{d\in[n]})\right\}_\lambda \approx \{\mathsf{Sim}(\mathcal{I},\ \{x_d\}_{d \text{ s.t. } C_d\in\mathcal{I}},\ y)\}_\lambda, \qquad y = \begin{cases} F(x_1, \cdots x_n) & O \in \mathcal{I} \\ \perp & O \notin \mathcal{I} \end{cases}$$

We use the notation (n, m, t)-MPC protocol for computing $\mathcal{F}$ to denote a n-client, m-server, t-secure MPC protocol for computing F. By default, we assume $(m-1)$-security and denote such protocols as (n, m)-MPC protcols.

Finally, we note that any secure m-client m-server protocol for F implies a standard m-party MPC for F by letting each party i simulate both client i and server i, and also act as the output client. Furthermore, it is without loss of generality to consider only Boolean, single-output, function $F : (\{0,1\}^*)^m \to \{0,1\}$, since in the semi-honest security setting, to compute a more general non-Boolean, mult-output, function $F' : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ for standard MPC protocols, the parties can execute in parallel different protocols for computing each bit in every output, where for every bit in the k'th output, only party k acts as the output client.

MPC with PKI setup. We consider both standard and client-server MPC protocols, with a public key infrastructure (PKI) setup. A PKI setup allows a one-time global choice of parameters $\mathsf{params} \leftarrow \mathsf{ParamGen}(1^\lambda)$, followed by independent choices of a key pair $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{params})$ by each party P_i.[5] We assume that each party knows the public keys of all parties with whom it wants to interact as well as its own secret key. Note that the public keys are generated independently of any inputs or even the number of other parties in the system. For this reason we do not count the PKI setup towards the round complexity of MPC protocols.

B Multi-Input HSS from LWE

We observe that general multi-input HSS for arbitrary polynomial-size circuits can be obtained from the Learning With Errors (LWE) assumption, by a simple variation of the 2-round MPC construction from spooky encryption of [DHRW16].

Recall an additive-function-sharing (AFS) spooky encryption scheme (Definition A.2) provides a procedure for k ciphertexts c_i of inputs x_i under k independent keys sk_i to be publicly transformed into k new ciphertexts c'_i, so that the resulting decrypted plaintext values (under the respective keys sk_i) form additive secret shares y_i of $C(x_1, \ldots, x_k)$. Dodis *et al.* [DHRW16] showed that AFS-spooky encryption for all polynomial-size circuits exists based on LWE.

Note that Dodis *et al.* [DHRW16] also observed that AFS-spooky encryption implies function secret sharing (FSS) for general circuits. As discussed in Section 3, FSS can be viewed as the special case of $(1, m)$-HSS (i.e., single-input HSS) for the universal function $F(x; P) = P(x)$. Taken as a

[5] We will only use params to specify a group for ElGamal encryption; hence, we can let params be a common random string, or even pick params deterministically under a suitable variant of DDH.

black box, FSS is weaker than general $(*, m)$-HSS in two ways: (1) it does not address the setting of multiple inputs (recall $(*, m)$ means arbitrarily many inputs); and (2) comparing within the single-input setting, HSS requires the share size of an input x to grow only as a function of the size of x, whereas the $(1, m)$-HSS resulting from an FSS scheme may have respective share size that grows with both x and the program P.

More directly relevant to us is the use of AFS-spooky encryption in [DHRW16] within the context of obtaining 2-round MPC. Essentially we observe that combining an additional layer on top of this construction enables HSS correctness and secrecy for multiple inputs.

Explicitly, consider the following (additive) $(*, m)$-HSS construction from a generic AFS-spooky encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SpookyEval})$.

$\mathsf{Share}_{\mathsf{HSS}}(1^\lambda, i, x_i)$: Client i performs the following to share input $x_i \in \{0,1\}^\ell$.

1. Sample m independent spooky encryption key pairs, $(\mathsf{pk}_{ij}, \mathsf{sk}_{ij}) \leftarrow \mathsf{Gen}(1^\lambda)$, for $j \in [m]$.
2. Additively secret share x_i into m shares over $\{0,1\}^\ell$: i.e., $(x_{i1}, \ldots, x_{im}) \leftarrow \mathsf{AdditiveShare}(x_i, m)$.
3. Encrypt each share x_{ij} under the jth encryption key: i.e., $c_{ij} \leftarrow \mathsf{Enc}(\mathsf{pk}_{ij}, x_{ij})\ \forall j \in [m]$.
4. For each $j \in [m]$, set the jth HSS share of x_i to be: $\mathsf{share}_i^j = \Big((\mathsf{pk}_{ij'})_{j'=1}^m, (c_{ij'})_{j'=1}^m, \mathsf{sk}_{ij}\Big)$.

$\mathsf{Eval}_{\mathsf{HSS}}(j, x_0, (\mathsf{share}_1^j, \ldots, \mathsf{share}_n^j))$: Server $j \in [m]$ performs the following.

1. For each $i \in [n]$, parse $\mathsf{share}_i^j = \Big((\mathsf{pk}_{ij'})_{j'=1}^m, (c_{ij'})_{j'=1}^m, \mathsf{sk}_{ij}\Big)$.
2. Perform spooky evaluation on *all* nm ciphertexts $(c_{ij'})_{i\in[n], j'\in[m]}$, for the function

$$F'((x_{ij'})_{i\in[n],j'\in[m]}) = F\Big(\bigoplus_{j'\in m} x_{1j'}, \ldots, \bigoplus_{j'\in[m]} x_{nj'}\Big).$$

   That is, let $(c'_{ij'})_{i\in[n],j'\in[m]} \leftarrow \mathsf{SpookyEval}(F, (\mathsf{pk}_{ij'}, c_{ij'})_{i\in[n],j'\in[m]})$.
3. For each $i \in [n]$, decrypt the corresponding ciphertext c_{ij} using key sk_{ij}: i.e., $y_i^j = \mathsf{Dec}(\mathsf{sk}_{ij}, c_{ij})$.
4. Output $y^j = \bigoplus_{i=1}^n y_i^j$ as the HSS-evaluated share.

$\mathsf{Dec}_{\mathsf{HSS}}(y^1, \ldots, y^m)$: Output $\bigoplus_{j=1}^m y^j$.

Theorem B.1. *Assuming the existence of AFS-spooky encryption for circuits, there exists $(*, m)$-HSS for circuits.*

Proof. Statistical correctness and security of the above construction follow directly by correctness of the spooky evaluation and semantic security of the AFS-spooky encryption scheme (in addition to privacy of the additive secret sharing scheme). In particular, note that each server has ability to decrypt only one *additive share* of each client's input. □

Plugging in the AFS-spooky encryption construction of [DHRW16], this yields $(*, m)$-HSS from LWE in the CRS model. From standard LWE, we obtain a form of *depth-dependent* $(*, m)$-HSS,

where the share size of an input x grows also with the depth of the circuits that can be homomorphically evaluated.[6] Further making a circular security assumption (see [Gen09, DHRW16]), we obtain an analogous result for standard HSS.

Corollary B.2 (Multi-input HSS with CRS setup from LWE)**.** *Assuming the hardness of α-DLWE for any $\alpha(\lambda) \in \lambda^{-\omega(1)}$, then for every polynomial $m(\lambda)$, there exists depth-dependent $(*, m)$-HSS for circuits with a CRS setup. Further making a circular security assumption [Gen09, DHRW16], there exists standard (non-leveled) $(*, m)$-HSS for circuits with a CRS setup.*

In the special case of single-client HSS, the need for CRS setup can be removed by having the client sample the CRS on his own:

Corollary B.3 (Single-input HSS from LWE)**.** *Assuming the hardness of α-DLWE for any $\alpha(\lambda) \in \lambda^{-\omega(1)}$, then there exists depth-dependent $(1, m)$-HSS for circuits. Further making a circular security assumption [Gen09, DHRW16], there exists standard (non-leveled) $(1, m)$-HSS for circuits.*

[6] This can be formalized by replacing the function $F(C; x_1, \ldots, x_n)$ of the HSS evaluation (as in Section 2.2) with the function $F'(C; (1^{d_1}, x_1), \ldots, (1^{d_n}, x_n))$, which outputs $\perp$ if the depth of C is greater than d_i for any $i \in [n]$.