Scooby: Improved Multi-Party Homomorphic Secret Sharing Based on FHE

Ilaria Chillotti[1], Emmanuela Orsini[2], Peter Scholl[3], Nigel Paul Smart[1,2], and Barry Van Leeuwen[2]

[1] Zama, Paris, France,
[2] imec-COSIC, KU Leuven, Leuven, Belgium,
[3] U. Aarhus, Aarhus, Denmark.
ilaria.chillotti@zama.ai,
emmanuela.orsini@kuleuven.be,
peter.scholl@cs.au.dk,
nigel.smart@kuleuven.be,
barry.vanleeuwen@kuleuven.be

Abstract. We present new constructions of multi-party homomorphic secret sharing (HSS) based on a new primitive that we call *homomorphic encryption with decryption to shares* (HEDS). Our first construction, which we call Scooby, is based on many popular fully homomorphic encryption (FHE) schemes with a linear decryption property. Scooby achieves an n-party HSS for general circuits with complexity $O(|F| + \log n)$, as opposed to $O(n^2 \cdot |F|)$ for the prior best construction based on multi-key FHE. Scooby can be based on (ring)-LWE with a super-polynomial modulus-to-noise ratio. In our second construction, Scrappy, assuming any generic FHE plus HSS for NC1-circuits, we obtain a HEDS scheme which does not require a super-polynomial modulus. While these schemes all require FHE, in another instantiation, Shaggy, we show how in some cases it is possible to obtain multi-party HSS without FHE, for a small number of parties and constant-degree polynomials. Finally, we show that our Scooby scheme can be adapted to use multi-key fully homomorphic encryption, giving more efficient spooky encryption and setup-free HSS. This latter scheme, Casper, if concretely instantiated with a B/FV-style multi-key FHE scheme, for functions F which do not require bootstrapping, gives an HSS complexity of $O(n \cdot |F| + n^2 \cdot \log n)$.

1 Introduction

One of the more interesting cryptographic constructions to be developed in recent years has been homomorphic secret sharing (HSS). This concept, which can be seen as a distributed analogue of homomorphic encryption, was introduced in [6], where a two party construction for branching programs was presented based on the decisional Diffie-Hellman assumption. The idea of HSS starts from the concept of a (traditional) secret sharing scheme, where an input x to some function is split into n shares, $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. This sharing, that in this work we always assume to be a full threshold sharing, is created via an algorithm $(\mathbf{x}_1, \ldots, \mathbf{x}_n) \leftarrow \mathsf{Share}^{\mathsf{HSS}}(x)$. An HSS scheme has two additional algorithms, the first $y_i \leftarrow \mathsf{Eval}^{\mathsf{HSS}}(F; \mathbf{x}_i)$ takes a function description F and a share $\mathbf{x}_i$ and produces a corresponding output share y_i. The second $\mathsf{Rec}^{\mathsf{HSS}}(y_1, \ldots, y_n)$ takes the output shares and reconstructs the result $F(x)$. To avoid trivial solutions one requires that the length of the y_j's should be *compact*, i.e. it only depends on the output length of the function F and the security parameter. An important class of HSS schemes are those with *additive reconstruction*, where the function $\mathsf{Rec}^{\mathsf{HSS}}$ simply computes $y_1 + \ldots + y_n$. We refer to these as *additive HSS schemes*. It is such additive HSS schemes that we focus on in this work.

Motivation for HSS. The main application of HSS is towards secure two-party or multi-party computation with succinct communication. Indeed, the breakthrough work of [6] showed that for a large class of circuits, it's possible to achieve secure computation with sublinear communication in the circuit size under DDH, which was previously only known using fully homomorphic encryption. Since then, HSS has proven useful in various other applications, and is closely related to pseudorandom correlation generators [3] and pseudorandom correlation functions [4], which allow generation of correlated randomness with a minimal amount of interaction. HSS for simple classes of functions, particularly the case of distributed point functions [23], has also proven useful for applications including private information retrieval [7] and secure RAM computation [19]. On a more theoretical side, HSS has also been used to build 2-round secure computation and nearly optimal worst-case to average-case reductions [8].

Additive reconstruction is an important feature of HSS in many secure computation settings, where it may be desirable for the output shares to be re-used in another secure computation based on secret sharing. This is the case, for instance, when using HSS to generate preprocessing material for multi-party computation protocols in the dishonest majority setting [3]. It can also be a useful feature in scenarios where a client reconstructing the output is constrained to perform only lightweight computations.

Current State of HSS and Related Primitives. Related to HSS is the dual concept of function secret sharing (FSS) [5, 7]. In FSS, the shared data is a secret function F (from some publicly known class of functions), such that the parties can locally obtain secret shares of $F(x)$, for any public input x. For general function classes such as polynomially-sized circuits, function secret sharing and homomorphic secret sharing are equivalent.

Obtaining efficient n-party HSS and FSS is complex for general functions. The most efficient known scheme is that based on an LWE-construction from spooky encryption. Spooky encryption, introduced by Dodis et al. [18], is a rather complex construction based on a multi-key variant of FHE [17, 25], and for our purposes we are only interested in *additive-function-sharing* spooky encryption (or AFS-spooky encryption). Spooky encryption is a semantically secure public-key encryption scheme consisting of the usual three algorithms $(\mathsf{KeyGen}^{\mathsf{Spooky}}, \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}}, \mathsf{Dec}^{\mathsf{Spooky}}_{\mathsf{sk}})$ as well as an additional algorithm

$\mathsf{Eval}^{\mathsf{Spooky}}_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}(F, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$. The $\mathsf{Eval}^{\mathsf{Spooky}}$ algorithm, given a function F on n arguments from a given class, and n ciphertexts ct_i, encrypting x_i under pk_i, produces n new ciphertexts $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n$ such that, computing $y_i \leftarrow \mathsf{Dec}^{\mathsf{Spooky}}_{\mathsf{sk}_i}(\mathsf{ct}_i)$, we have that $y_1 + \ldots + y_n = F(x_1, \ldots, x_n)$.

In [18], it is shown that it is possible to build FSS from AFS-spooky encryption. Roughly, to *share* an input function F the dealer first generates n AFS-spooky key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}^{\mathsf{Spooky}}(1^\lambda)$. The dealer also generates an n-out-of-n description of the function F, i.e. functions $F_i(x)$ such that $F(x) = F_1(x) + \ldots + F_n(x)$. Finally, the function secret sharing of the input function F is defined to be the tuple $F_i = (\mathsf{sk}_i, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}_1}(F_1), \ldots, \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}_n}(F_n))$.

To define the FSS evaluation we create a function C_x which takes as input the n additive shares of a function F, and evaluates it on the input x, which is hard-coded into C_x. By applying

$$\mathsf{Eval}^{\mathsf{Spooky}}_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}\left(C_x, \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}_1}(F_1), \ldots, \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}_n}(F_n)\right),$$

we obtain ciphertexts $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n$, where ct'_i can be decrypted (using sk_i) to obtain y_i such that $y_1 + \ldots + y_n = F(x)$.

In [8], Boyle et al. showed how the FSS construction from spooky encryption can be modified to enable an additive HSS scheme. The $\mathsf{Share}^{\mathsf{HSS}}(x)$ operation additively shares x into $x = x_1 + \ldots + x_n$, generates n spooky key pairs $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}^{\mathsf{Spooky}}(1^\lambda)$, and then encrypts x_i via $\mathsf{ct}_i \leftarrow \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}_i}(x_i)$. The share values $\mathbf{x}_i$ output by $\mathsf{Share}^{\mathsf{HSS}}(x)$ being $\mathbf{x}_i = (\{\mathsf{pk}_i\}_{i=1}^n, \{\mathsf{ct}_i\}_{i=1}^n, \mathsf{sk}_i)$. The $\mathsf{Eval}^{\mathsf{HSS}}(F, \mathbf{x}_i)$ function executes $\mathsf{Eval}^{\mathsf{Spooky}}_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}$ on the function F and the ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ so as to obtain n ciphertexts $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n$. The output of $\mathsf{Eval}^{\mathsf{HSS}}(F, \mathbf{x}_i)$ then being $\mathsf{Dec}^{\mathsf{Spooky}}_{\mathsf{sk}_i}(\mathsf{ct}'_i)$.

Thus, there is a strong connection between HSS, FSS and spooky constructions, and, as mentioned above, the prior most efficient n-party HSS and FSS constructions for circuits arise from AFS-spooky based on LWE (and a circular security assumption). The best current construction for AFS-spooky encryption of Dodis et al. [18] has a complexity of $O(n^2 \cdot |F|)$. In particular, each gate of the underlying arithmetic circuit F requires a bootstrapping operation which in the multi-key FHE setting has complexity $O(n^2)$.

1.1 Our Contribution

We present new constructions of homomorphic secret sharing in the multi-party setting, supporting up to $n-1$ out of n corruptions. Our constructions improve upon the only previous general construction, based on AFS-spooky encryption [18], either by being more efficient, or in some cases, relying on different assumptions.

HSS from Homomorphic Encryption with Decryption Shares. We present our constructions as a new primitive called *homomorphic encryption with decryption to shares (HEDS)*, which can be seen as a homomorphic encryption scheme with a special decryption algorithm that (non-interactively) outputs an n-party secret share of the encrypted message. HEDS is closely related to both spooky encryption and homomorphic secret sharing (HSS): the major difference compared to spooky is that HEDS needs to set up private decryption keys under a common public key with either a trusted setup algorithm or a secure multiparty computation protocol, while the difference with HSS is that the homomorphic evaluation algorithm is public. As is the case for spooky, HEDS immediately implies additive HSS for the same class of functions.

Scooby Construction: HEDS from Linear Decryption FHE. We show that HEDS can be built using any FHE scheme with a special decryption property, which we call *linear decryption based fully homomorphic encryption* (LD-based FHE) schemes. Examples of such LD-based FHE schemes are LWE-based constructions like BGV [11], BFV [20], GSW [22] and TFHE [15, 16]. Notice this special property of almost all FHE schemes, where the decryption function is a linear function of the secret key, has been exploited previously, including for HSS in the two-party setting [18, 9] and other applications [10, 21].

Any of these schemes can be used to instantiate our Scooby construction, giving additive HSS for circuits. Recall in AFS-spooky the key generation is run independently by the n-parties, in our variation the keys are instead generated by a trusted third party.[4]

Since this construction only requires single-key FHE and not multi-key FHE, we obtain n-party HSS that is simpler and more efficient than the AFS-spooky-based construction. In particular, the computational complexity grows as $O(|F| + \log n)$, whereas AFS-spooky has complexity $O(n^2 \cdot |F|)$ for n parties. In addition, when instantiated with BGV we show that the standard parameter sets for bootstrapping are sufficient for our construction.

At a high level, at the core of Scooby is a well-known 2-party distributed decryption procedure, which non-interactively decrypts an LWE-based ciphertext into two shares, assuming the ciphertext modulus has super-polynomial size. This trick has been used previously, including in the construction of AFS-spooky. Our main contribution is to bootstrap this 2-party non-interactive algorithm into an n-party non-interactive algorithm. We do this by placing the n parties on the leaves of a binary tree, and then homomorphically evaluating the two party protocol at each internal node of the tree. Each party only needs to evaluate the 2-party protocol at each node on the path from the root to its leaf. Each homomorphic evaluation at the internal nodes is exactly equivalent to a bootstrapping operation, namely a homomorphic evaluation of the decryption circuit for some key. Thus, decryption into shares costs $O(\log n)$ operations per party.

Removing the Super-polynomial Modulus. The problem with Scooby, as well as all LWE-based additive HSS schemes, is that we require a super-polynomial modulus-to-noise ratio in the underlying LD-based FHE scheme. This is a stronger form of LWE assumption that usually requires larger pa-

[4] In some sense the "spooky" behaviour exhibited by spooky encryption cannot really be explained, whereas our "spooky" behaviour can be explained by the setup procedure. This setup procedure in some sense acts like the janitor in Scooby-Doo, who has set up the spooky goings-on.

Construction	Assumptions	Setup	Complexity
DHRW [18] (AFS-Spooky)	LWE with super-polynomial modulus	Uniform CRS	$O(n^2 \cdot \lvert F \rvert)$
Scooby: §5 (HEDS)	LD-based FHE with super-polynomial modulus	Trusted	$O(\lvert F \rvert + \log n)$
Scrappy: §6.1 (HEDS)	Generic FHE + 2-party HSS for NC1	Trusted	$O(\lvert F \rvert + \log n)$
Shaggy: §6.2 (HEDS)	2-party HSS for NC1	Trusted	$O(1)$ ($n = 4$, constant-deg F)
Casper: (Full Version) (AFS-Spooky)	Specific MK-FHE with super-polynomial modulus	Uniform CRS	$O(n \cdot \lvert F \rvert + n^2 \cdot \log n)$ or $O(n^2 \cdot \lvert F \rvert + n^2 \cdot \log n)$

Table 1. Summary of n-party HSS Constructions. All FHE-based constructions allow arbitrary functions F, and assume circular security to avoid blow-up in the key sizes (this assumption can be removed by relaxing to bounded-depth circuits). The asymptotic complexities ignore potential factors in λ that are independent of n and F.

rameters to compensate. We give a variant of the construction where we only need standard FHE, together with an HSS scheme for NC1 circuits. Using recent constructions of HSS [27, 29, 1] based on either Paillier encryption or class groups, we obtain the first additive HSS schemes for circuits that do not require LWE with a super-polynomial modulus. The complexity of the HSS is also $O(|F| + \log n)$, however, it is likely to be less efficient in practice than Scooby. We call this construction Scrappy. We summarize our results in Table 1.

Avoiding FHE Entirely. We also show that in certain cases, we can obtain multi-party HSS without using any form of FHE whatsoever. We do this through a variant of the previous construction, where we bootstrap a HEDS scheme to handle more parties by homomorphically evaluating its own decryption circuit. This transformation is more challening to apply without resorting to FHE, and we are only able to obtain a 4-party HEDS scheme for constant-degree polynomials, based on Paillier encryption. Nevertheless, as far as we are aware, this is the first instance of $>$ 2-party, dishonest majority HSS for constant-degree polynomials, without relying on FHE. We call this construction Shaggy.

Spooky from HEDS. In addition, in the full version, we show how our Scooby scheme can be adapted to give a true AFS-spooky encryption, i.e. with no trusted setup and independent keys, if we base our construction on *specific* multi-key FHE (MK-FHE). This instantiation can have a simpler complexity than that given in [18], in particular, assuming the function F can be evaluated without bootstrapping, our complexity is $O(n \cdot |F| + n^2 \cdot \log n)$. If F requires a bootstrapping for all the operations, it is $O(n^2 \cdot |F| + n^2 \cdot \log n)$. We call this construction Casper.

We give two variants of Casper, one based on the TFHE scheme [13], and one based on the BFV scheme [14]. We note that being MK-FHE schemes, the construction will be less efficient than our Scooby scheme, which works over most (practical) FHE schemes. It is interesting to note that the spooky construction from [18] also goes via MK-FHE. In particular, they make use of the MK-FHE scheme of [17, 25]. The route though is more complex than our tree-based construction, leading to an increased complexity.

2 Preliminaries

For a set S, we denote by $a \leftarrow S$ the process of drawing a from S with a uniform distribution on the set S. If D is a probability distribution, we denote by $a \leftarrow D$ the process of drawing a with the given probability distribution. For a probabilistic algorithm A, we denote by $a \leftarrow A$ the process of assigning a the output of algorithm A; with the underlying probability distribution being determined by the random coins of A.

All reductions modulo an integer p will be assumed to be centred, i.e. in the interval $(-p/2, \ldots, p/2)$.

We let $R = \mathbb{Z}[X]/(X^N + 1)$ and R_p denote the localisation of R at p, i.e. $(\mathbb{Z}/p\mathbb{Z})[X]/(X^N + 1)$. For a real interval I we let R_I denote the restriction of the *set* R to have coefficients in the support of I. Thus as sets (but not as rings) we have $R_q = R_{(-q/2,\ldots,q/2)}$.

2.1 Homomorphic Secret Sharing

The following definition of public-key HSS is adapted from [9]. Note that, as we are only interested in schemes with additive reconstruction, we can disregard the decoding algorithm, $\mathsf{Dec}^{\mathsf{HSS}}_{\mathsf{sk}}$, that is given in the more general definition of HSS [8]. Concretely, in additive HSS the decoding algorithm simply adds up all the shares.

Definition 2.1 (Additive Public Key Homomorphic Secret Sharing). *An n-party, public-key* homomorphic secret sharing (HSS) *scheme for a class of functions $\mathcal{F}$ over a ring $\mathcal{R}$ with input space $\mathcal{I} \subseteq R$ consists of PPT algorithms* $(\mathsf{KeyGen}^{\mathsf{HSS}}, \mathsf{Share}^{\mathsf{HSS}}_{\mathsf{pk}}, \mathsf{Eval}^{\mathsf{HSS}}_{\mathsf{pk}})$ *with the following syntax:*

- $\mathsf{KeyGen}^{\mathsf{HSS}}(1^\lambda, n) \rightarrow (\mathsf{pk}, (\mathsf{ek}_1, \ldots, \mathsf{ek}_n))$*: Given a security parameter* 1^λ*, the setup algorithm outputs a public key* pk *and n evaluation keys* $(\mathsf{ek}_1, \ldots, \mathsf{ek}_n)$*.*
- $\mathsf{Share}^{\mathsf{HSS}}_{\mathsf{pk}}(\mathsf{pk}, x) \rightarrow (\mathbf{x}_1, \ldots, \mathbf{x}_n)$*: Given public key* pk *and private input value $x \in \mathcal{I}$, the share algorithm outputs shares* $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$*.*
- $\mathsf{Eval}^{\mathsf{HSS}}_{\mathsf{pk}}(F; \mathbf{x}_i, \mathsf{ek}_i) \rightarrow y_i$*: On input a function $F \in \mathcal{F}$, the parties share $\mathbf{x}_i$, and it's evaluation key ek_i, the homomorphic evaluation algorithm outputs $y_i \in R$, which is party i's share of an output $y \in R$.*

This definition is in the multi-input setting, meaning that it supports a compact evaluation of a function F on shares of inputs $x^{(1)}, \ldots, x^{(\rho)}$ given by ρ parties that are usually referred to as *clients*. More concretely, each client inputs $x^{(i)}$ to the Share algorithm which returns shares $\mathbf{x}^{(i)}_j, j \in [n]$, to n parties (the *servers*). Each server can then locally run Eval on input $(\mathbf{x}^{(1)}_j, \ldots, \mathbf{x}^{(\rho)}_j)$ obtaining a share y_j such that $F(x^{(1)}, \ldots, x^{(\rho)}) = \sum_{j \in [n]} y_j$. Note that the $\mathsf{KeyGen}^{\mathsf{HSS}}$ algorithm cannot be run by any single party, so can be seen as a form of correlated randomness generated by a trusted dealer. We describe the required security properties for the algorithms $(\mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ according to this more general formulation.

Security Experiment $\mathsf{Exp}^{\mathsf{HSS,sec}}_{\mathcal{A},j}(\lambda)$

Let $I \subset [n]$ be the set of corrupt servers.

1. $(\mathsf{pk}, (\mathsf{ek}_1, \ldots, \mathsf{ek}_n)) \leftarrow \mathsf{KeyGen}^{\mathsf{HSS}}(1^\lambda)$.
2. $(x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$.
3. $b \leftarrow \{0, 1\}$.
4. $(\mathbf{x}_{b,1}, \ldots, \mathbf{x}_{b,n}) \leftarrow \mathsf{Share}^{\mathsf{HSS}}_{\mathsf{pk}}(\mathsf{pk}, x_b)$.
5. $b' \leftarrow \mathcal{A}(\mathsf{state}, \mathsf{pk}, \{\mathsf{ek}_j, \mathbf{x}_{b,j}\}_{j\in I})$.
6. Return $b' = b$.

Fig. 1. Security Experiment $\mathsf{Exp}^{\mathsf{HSS,sec}}_{\mathcal{A},j}(\lambda)$

Definition 2.2 (HSS (Statistical) Correctness). *We say that an n-party public-key HSS scheme $(\mathsf{KeyGen}^{\mathsf{HSS}}, \mathsf{Share}^{\mathsf{HSS}}_{\mathsf{pk}}, \mathsf{Eval}^{\mathsf{HSS}}_{\mathsf{pk}})$ is correct for a class of functions $\mathcal{F}$ if, for all security parameters $\lambda \in \mathbb{N}$, for all functions $F \in \mathcal{F}$, for all $x^{(1)}, \ldots, x^{(\rho)} \in \mathcal{I}$ (where $\mathcal{I}$ is the input space of F), for all $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_n) \leftarrow \mathsf{KeyGen}^{\mathsf{HSS}}(1^\lambda)$ and for all $(\mathbf{x}^{(i)}_1, \ldots, \mathbf{x}^{(i)}_n) \leftarrow \mathsf{Share}^{\mathsf{HSS}}_{\mathsf{pk}}(\mathsf{pk}, x^{(i)}), i \in [\rho]$, we have*

$$\Pr\left[y_1 + \cdots + y_n = F(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda),$$

where

$$y_j \leftarrow \mathsf{Eval}^{\mathsf{HSS}}_{\mathsf{pk}}(F; (\mathbf{x}^{(1)}_j, \ldots, \mathbf{x}^{(\rho)}_j), \mathsf{ek}_j),\ j \in [n],$$

where the probability is taken over the random coins of $\mathsf{KeyGen}^{\mathsf{HSS}}$, $\mathsf{Share}^{\mathsf{HSS}}_{\mathsf{pk}}$ and $\mathsf{Eval}^{\mathsf{HSS}}_{\mathsf{pk}}$.

Definition 2.3 (HSS Security). *Let I be the set of corrupt servers. For each $j \in I$ and non-uniform adversary $\mathcal{A}$ (of size polynomial in the security parameter λ), it holds that*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{HSS,sec}}_{\mathcal{A},j}(\lambda) = 1]\right| \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{HSS,sec}}_{\mathcal{A},j}(\lambda)$ is the experiment defined in Figure 1.

Remark 2.1 (Private-key HSS). HSS can also be defined in the single-input, private key setting, which is weaker than the public-key flavour above. Here, there is no KeyGen algorithm, and Share is run only once on all inputs together, so can be seen as a trusted dealer algorithm that distributes the shares.

2.2 Spooky Encryption

"Spooky" encryption is a type of public key encryption scheme which exhibits a form of limited malleability, so called "spooky action at a distance" [18]. The particular form of spooky encryption we will use is so called *additive-function-sharing* spooky encryption (or AFS-spooky encryption). We present a definition which works for any finite ring $\mathcal{R}$, and arithmetic circuit C, and not just for the case of $\mathbb{F}_2$ as originally presented.

Definition 2.4 (AFS-spooky Encryption). *An* AFS-spooky *encryption scheme, over a finite field $\mathbb{F}_p$, is a public-key encryption scheme given by a tuple of four algorithms* $(\mathsf{KeyGen}^{\mathsf{Spooky}}, \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}}, \mathsf{Dec}^{\mathsf{Spooky}}_{\mathsf{sk}}, \mathsf{Eval}^{\mathsf{Spooky}}_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n})$ *with the following syntax:*

- $\mathsf{KeyGen}^{\mathsf{Spooky}}(1^\lambda)$*: This is a probabilistic polynomial time algorithm which on input of a security parameter λ outputs a public/private key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}}(m)$*: This probabilistic polynomial time algorithm takes a message $m \in \mathcal{R}$ and generates a ciphertext* ct *encrypting that message under the public key* pk.
- $\mathsf{Dec}^{\mathsf{Spooky}}_{\mathsf{sk}}(\mathsf{ct})$*: Given a ciphertext* ct *encrypted under the public key associated to* sk*, this algorithm produces the underlying plaintext.*
- $\mathsf{Eval}^{\mathsf{Spooky}}_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}(C, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$*: Given an arithmetic circuit description $C : \mathcal{R}^n \longrightarrow \mathcal{R}$, n public keys* $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$*, and n of ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$*, this produces n ciphertexts* $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n$

An AFS-spooky encryption scheme must be correct, as an encryption scheme, i.e. we must have

$$\forall(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}^{\mathsf{Spooky}}(1^\lambda),\ \forall m \in \mathcal{R} \quad : \quad \mathsf{Dec}^{\mathsf{Spooky}}_{\mathsf{sk}}(\ \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}}(m)\ ) = m.$$

It must also be IND-CPA as an encryption scheme and satisfy the following form of limited malleability called AFS-spooky correctness.

Definition 2.5 (AFS-spooky Correctness). *There exists a negligible function ν such that for all $\lambda \in \mathbb{N}$, every arithmetic circuit C computing a n-argument function $f : \mathcal{R}^n \longrightarrow \mathcal{R}$, and all inputs $x_1, \ldots, x_n$ of C, we have*

$$\Pr\left[\sum_{i\in[n]} y_i = C(x_1, \ldots, x_n) : \begin{array}{l} \forall i \in [n], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}^{\mathsf{Spooky}}(1^\lambda), \\ \forall i \in [n], \mathsf{ct}_i \leftarrow \mathsf{Enc}^{\mathsf{Spooky}}_{\mathsf{pk}}(x_i), \\ (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n) \leftarrow \mathsf{Eval}^{\mathsf{Spooky}}_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}(C, \mathsf{ct}_1, \ldots, \mathsf{ct}_n), \\ \forall i \in [n], y_i \leftarrow \mathsf{Dec}^{\mathsf{Spooky}}_{\mathsf{sk}_i}(\mathsf{ct}'_i) \end{array}\right] \geq 1-\nu(\lambda)$$

In [18], it is shown how to construct an AFS-spooky encryption scheme in the CRS model using an LWE-based multi-key FHE [17, 25] and assuming a circular security assumption. The common reference string (output by a separate generation algorithm), necessary in the multi-key FHE construction, is assumed as input to the key generation algorithm, and correctness and security hold for all outputs of the common reference string generator.

In their work, Dodis et al. [18] show that AFS-spooky encryption implies FSS for general circuit; in [8], Boyle et al. show that AFS-spooky also enables HSS for multiple inputs; in fact, it implies HSS without any setup, where the key generation algorithm is simply run locally by each client providing input.

3 Homomorphic Encryption with Decryption to Shares (HEDS)

In this section we formally introduce the notion of a scheme which implements Homomorphic Encryption with Decryption to Shares (HEDS) and relate it with other concepts described in previous sections. Loosely speaking, a HEDS encryption scheme is similar to public-key HSS, except with a public evaluation algorithm that outputs a ciphertext, more akin to evaluation in homomorphic encryption. The ciphertext is then convert into shares in the decryption algorithm, which uses one party's private key. In addition, similarly to HSS, but unlike in spooky encryption, the parties need to engage in a protocol, or assume a trusted third party, to set up the associated public and secret keys. Thus the action from the outside seems spooky, but this can be explained away as an effect of the setup protocol.

We start by giving the definition of HEDS, and then we show that it enables both homomorphic and function secret sharing.

Definition 3.1 (HEDS Encryption). *A* HEDS encryption scheme *for a class of functions $\mathcal{F} : \mathcal{R}^* \to \mathcal{R}$, over a ring $\mathcal{R}$, is given by a tuple of PPT algorithms* $(\mathsf{SetUp}^{\mathsf{HEDS}}, \mathsf{Enc}^{\mathsf{HEDS}}_{\mathsf{pk}}, \mathsf{Dec}^{\mathsf{HEDS}}_{\mathsf{sk}}, \mathsf{Eval}^{\mathsf{HEDS}}_{\mathsf{pk}})$*, with the following syntax:*

- $\mathsf{SetUp}^{\mathsf{HEDS}}(1^\lambda, n)$*: This randomized algorithm takes as input a security parameter λ, a number of parties n. It outputs the tuple* $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
- $\mathsf{Enc}^{\mathsf{HEDS}}_{\mathsf{pk}}(m)$*: This takes as input the public key and a message $m \in \mathcal{R}$, and outputs a ciphertext* ct.
- $\mathsf{Dec}^{\mathsf{HEDS}}_{\mathsf{sk}_i}(\mathsf{ct})$*: Given a ciphertext* ct *encrypted under the public key this outputs a value y_i for each $i \in [n]$.*
- $\mathsf{Eval}^{\mathsf{HEDS}}_{\mathsf{pk}}(C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\rho))$*: On input of the public key* pk*, a set of n ciphertexts, and an arithmetic circuit description $C : \mathcal{R}^\rho \longrightarrow \mathcal{R}$ of a function from the specified class, this produces a ciphertext* ct.

The algorithms $\left(\mathsf{SetUp}^{\mathsf{HEDS}}, \mathsf{Enc}^{\mathsf{HEDS}}_{\mathsf{pk}}, \mathsf{Dec}^{\mathsf{HEDS}}_{\mathsf{sk}}, \mathsf{Eval}^{\mathsf{HEDS}}_{\mathsf{pk}}\right)$ should satisfy the following correctness and security requirements.

Definition 3.2 (HEDS Correctness). *There exists a negligible function ν such that for all $\lambda \in \mathbb{N}$, every arithmetic circuit C computing a ρ-argument function $f : \mathcal{R}^\rho \longrightarrow \mathcal{R}$ in $\mathcal{F}$, and all inputs $x_1, \ldots, x_\rho$ of C, we have*

$$\Pr\left[\textstyle\sum_{i\in[n]} y_i = C(x_1, \ldots, x_\rho) \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{SetUp}^{\mathsf{HEDS}}(1^\lambda, n), \\ \forall i \in [\rho], \mathsf{ct}_i \leftarrow \mathsf{Enc}^{\mathsf{HEDS}}_{\mathsf{pk}}(x_i), \\ \mathsf{ct} \leftarrow \mathsf{Eval}^{\mathsf{HEDS}}_{\mathsf{pk}}(C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\rho)), \\ \forall i \in [n], y_i \leftarrow \mathsf{Dec}^{\mathsf{HEDS}}_{\mathsf{sk}_i}(\mathsf{ct}) \end{array}\right] \geq 1 - \nu(\lambda).$$

Definition 3.3 (HEDS Security). *For all subsets $A \subset [n]$ of size $< n$, and all probabilistic polynomial time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ we have*

$$\Pr\left[b = b' \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{SetUp}^{\mathsf{HEDS}}(1^\lambda, n), b \in \{0,1\}, \\ (m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in A}), \\ \mathsf{ct} \leftarrow \mathsf{Enc}^{\mathsf{HEDS}}_{\mathsf{pk}}(m_b), \\ b' \leftarrow \mathcal{A}_2(\mathsf{ct}, \mathsf{state}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

i.e. the encryption scheme is IND-CPA, even when up to $n-1$ secret keys are given to the adversary.

Compactness. Just as with fully homomorphic encryption, we say that HEDS is *compact* if the share decryption algorithm is independent of the evaluated function.

3.1 Multi-input HSS from HEDS Encryption

Here we relate HEDS encryption and HSS showing that HEDS encryption implies HSS with multiple inputs. Let $\mathcal{P}$ be a set of n servers and $\mathcal{C}$ be a set of m clients. Let C be a circuit representing a function $F : \mathcal{R}^m \to \mathcal{R}$ in a class function $\mathcal{F}$. To build an HSS-scheme, we need to define three algorithms $\mathsf{KeyGen}^{\mathsf{HSS}}$, $\mathsf{Share}^{\mathsf{HSS}}$, $\mathsf{Eval}^{\mathsf{HSS}}$ as in Definition 2.1. Let $(\mathsf{SetUp}^{\mathsf{HEDS}}, \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{HEDS}}$, $\mathsf{Dec}_{\mathsf{sk}}^{\mathsf{HEDS}}$, $\mathsf{Eval}_{\mathsf{pk}}^{\mathsf{HEDS}})$ be a HEDS encryption scheme for $\mathcal{F}$, as defined in the previous section, we proceed as follows.

- $\mathsf{KeyGen}^{\mathsf{HSS}}(1^\lambda, n)$:
  1. Run $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{SetUp}^{\mathsf{HEDS}}(1^\lambda, n)$
  2. For each $i \in [n]$, set $\mathsf{ek}_i := \mathsf{sk}_i$
  3. Return pk and $(\mathsf{ek}_1, \ldots, \mathsf{ek}_n)$
- $\mathsf{Share}_{\mathsf{pk}}^{\mathsf{HSS}}(x^{(j)})$: Each client $P_j \in \mathcal{P}$, on input $x^{(j)}$ performs the following steps. We recall that the goal is to obtain shares $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of $(x^1, \ldots, x^m)$.
  1. For $i \in [n]$ and $j \in [m]$, generate $x_i^{(j)}$ such that $x^{(j)} = x_1^{(j)} + \ldots + x_n^{(j)}$.
  2. For each $x_i^{(j)}$, compute $\mathsf{ct}_i^{(j)} = \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{HEDS}}(x_i^{(j)})$.
  3. Set $\mathbf{x}_i = \{\mathsf{ct}_i^{(j)}\}_{j \in [m]}$, for $i \in [n]$.
- $\mathsf{Eval}_{\mathsf{pk}}^{\mathsf{HSS}}(F; \mathbf{x}_i, \mathsf{ek}_i)$: Given a function $F : \mathcal{R}^m \to \mathcal{R}$, each server $i \in [n]$ computes circuit description C of F and proceeds as follows.
  1. Compute $\mathsf{ct}_i = \mathsf{Eval}_{\mathsf{pk}}^{\mathsf{HEDS}}\left(C, (\mathsf{ct}_i^{(1)}, \ldots, \mathsf{ct}_i^{(m)})\right)$
  2. Compute $y_i = \mathsf{Dec}_{\mathsf{ek}_i}^{\mathsf{HEDS}}(\mathsf{ct}_i)$

By Definition 3.2, we know that the evaluation algorithm outputs to the servers the shares $y_1, \ldots, y_n$ such that $\sum_{i \in [n]} y_i = y = F(x^{(1)}, \ldots, x^{(m)})$.

Proposition 3.1. *Assuming the existence of a HEDS encryption scheme for a class of functions $\mathcal{F}$, there exists a public-key multi-input HSS scheme for $\mathcal{F}$.*

Proof. Correctness follows by inspection of the scheme described above and by correctness of the underlying HEDS construction. Security also follows from the security of HEDS. □

In the other direction, we observe that a public-key HSS scheme implies HEDS for the same class of functions, however, the resulting HEDS scheme may not be compact. This is because the HSS evaluation algorithm will have to be carried out in the HEDS decryption step, since HSS uses a private key for evaluation.

4 Linear-Decryption Based FHE

Our main constructions are based on a form of FHE which comes from LWE-style systems. We abstract much of the details of the specific construction away in what follows, for example the specific key generation and encryption algorithms. This allows us to capture schemes as diverse as BGV [11], BFV [20], GSW [22] and TFHE [15, 16]. These schemes all have the same form of decryption equation, namely one based on a linear inner product combination of the ciphertext with the secret key, modulo the ciphertext modulus. The result of this inner product is then processed to produce the plaintext (which is an element of R_p for some prime p) in one of two distinct ways, depending on whether the message is embedded at the top of the range modulo q (as in FV), or the bottom of the range modulo q (as in BGV). We refer to these two types of decryption as FHE as being of type msb and type lsb respectively. We call the whole class of such FHE systems Linear Decryption based, or LD-based FHE. Similar definitions have been considered previously [9, 21, 10].

Let sec denote some statistical security parameter and λ denote a computational security parameter. We define such a scheme as follows, the precise encryption and evaluation algorithms are not important for our discussion.

Definition 4.1 (LD-based FHE). *An* LD-based FHE *scheme is given by a tuple of algorithms* $(\mathsf{KeyGen}^{\mathsf{FHE}}, \mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}}, \mathsf{Dec}^{\mathsf{FHE}}_{\mathsf{sk}}, \mathsf{Eval}^{\mathsf{FHE}}_{\mathsf{pk}})$*, as follows:*

- $\mathsf{KeyGen}^{\mathsf{FHE}}(1^\lambda, p)$*: This randomized algorithm takes as input the security parameter* λ *and a plaintext modulus space* p*. It outputs a tuple* $(q, N, B, d, \Delta, S, \mathsf{pk}, \mathsf{sk})$*. The value* q *will correspond to the ciphertext modulus*[5]*, the value* N *will be the LWE-ring dimension (which for convenience we assume is a power of two), the value* B *will be a "noise bound", the value* d *is one less than the dimension of the ciphertext space, the value* Δ *is set to be* $\lfloor q/p \rfloor$*, the value* S *is a bound on the secret key size* S*, and* pk *(resp.* sk*) will be the public (resp. private) keys.*
  The private key $\mathsf{sk} = (s_1, \ldots, s_d)$ *is assumed to be a random element in* R_q^d *sampled such that* $\|\mathsf{sk}\|_\infty \le S$*. Note, this is not necessarily sampled uniformly at random subject to this constraint.*
  All subsequent algorithms are assumed to take the tuple (N, q, d, B, Δ) *implicitly as input parameters.*
- $\mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}}(m, \mathsf{type})$*: On input of* $m \in R_p$ *this will output a ciphertext* $\mathsf{ct} \in R_q^{d+1}$ *such that*

$$\mathsf{ct} \cdot (1, -\mathsf{sk}) = \begin{cases} m + p \cdot \epsilon \pmod{q} & \textit{If } \mathsf{type} = \mathsf{lsb}, \\ \Delta \cdot m + \epsilon \pmod{q} & \textit{If } \mathsf{type} = \mathsf{msb}. \end{cases}$$

A ciphertext such that $\|\epsilon\|_\infty \le B$ *will be called* valid*. The encryption algorithm produces such a valid ciphertext. The precise algorithm use for encryp-*

[5] In practice there may be many ciphertext moduli depending on which level a ciphertext is sitting at, at a high level this can be ignored. Although it can be important in practice

tion will depend on the public key, and the specific scheme. All that concerns us is the form of the ciphertext.

– $\mathsf{Eval}^{\mathsf{FHE}}_{\mathsf{pk}}(F(x_1,\ldots,x_\ell),\{\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell\})$*: On input of ℓ valid ciphertexts ct_i and an arithmetic function $F(x_1,\ldots,x_\ell)$ this function will homomorphically evaluate the function F over the ciphertexts, producing a valid ciphertext as output.*
– $\mathsf{Dec}^{\mathsf{FHE}}_{\mathsf{sk}}(\mathsf{ct})$*: On input of a valid ciphertext and a secret key this will compute the message as*

$$m = \begin{cases} (\mathsf{ct}\cdot(1,-\mathsf{sk}) \pmod{q}) \pmod{p} & \textit{If } \mathsf{type}=\mathsf{lsb},\\ \left\lfloor (\mathsf{ct}\cdot(1,-\mathsf{sk}) \pmod{q})\cdot p/q \right\rceil & \textit{If } \mathsf{type}=\mathsf{msb}. \end{cases}$$

The correctness requirement simply says that $\mathsf{Eval}^{\mathsf{FHE}}$, when given ℓ valid ciphertexts, outputs a valid encryption of the correct result. The security requirement is the standard notion of IND-CPA security.

For example: In the case of the BGV scheme [11] from ring-LWE we will have that $\mathsf{ct}=(c_0,c_1)$, so that decryption is given by $\mathsf{ct}\cdot(1,-\mathsf{sk})=c_0-s_1\cdot c_1$, and, hence, for this scheme we have $n=1$ and $\mathsf{sk}=s_1$. The BFV scheme [20] has the same structure, the main difference being that BFV uses the msb decryption, while BGV uses lsb.

In the case of Ring-GSW, a ciphertext is in $R_q^{(d+1)\times(d+1)\ell}$, with $d=1$ and $\mathsf{sk}=s_1$. In practice it is composed by $2\cdot\ell$ FV-like ciphertext (i.e., with the message encrypted in the msb). To decrypt a Ring-GSW ciphertext, we only decrypt one of these ciphertexts: the others contain redundant information. Another way of seeing a Ring-GSW ciphertext, is with a very sparse secret key $\mathsf{sk}=(\mathsf{sk}_1,\ldots,\mathsf{sk}_{2\ell})$, where all the keys corresponding to the FV-like ciphertext that we are not going to decrypt are set to zero. The TFHE scheme [15, 16] uses a combination of FV-like ciphertexts (with message encrypted in the msb, called LWE and RLWE ciphertexts) and Ring-GSW ones.

Parameters for Decryption to Shares. For such LD-based FHE schemes we have a special form of non-interactive two party distributed decryption, which we shall now outline in the lsb and the msb cases. We will require the parameters are selected so that

$$q > 2\cdot p\cdot(B+1)\cdot 2^{\mathsf{sec}}, \tag{1}$$

where sec is the statistical security parameter. This two-party distributed decryption, which is essentially the same technique as in [18, 9], will form the basis of our first multi-party HEDS construction in Section 5.

4.1 Two-Party Distributed Decryption: Type lsb

Suppose sk is split into two keys sk_1 and sk_2 with $\mathsf{sk}=\mathsf{sk}_1+\mathsf{sk}_2 \pmod{q}$, with sk_1 held by party P_1 and sk_2 held by party P_2. Now we can, without interaction, given a valid ciphertext ct encrypting a message m, compute an additive sharing of $m=m_1+m_2 \pmod{p}$ between P_1 and P_2 as follows. We require that the parties have agreed upon a public random value for each decryption, but later will remove this using a PRF.

2-party $\mathsf{DistDec}^{\mathsf{lsb}}$: Let $R \leftarrow \mathbb{Z}_q$ be a public random nonce.

1. P_1 computes $d_1 \leftarrow \mathsf{ct}\cdot(1, -\mathsf{sk}_1)+R \pmod q$ and then $m_1 \leftarrow d_1 \pmod p$.
2. P_2 computes $d_2 \leftarrow \mathsf{ct}\cdot(0, -\mathsf{sk}_2)-R \pmod q$ and then $m_2 \leftarrow d_2 \pmod p$.

We prove that this leads to a correct result with overwhelming probability.

Proposition 4.1. *Given an LD-based FHE scheme of type* msb *(Definition 4.1), where* $(q, N, B, d, \Delta, S, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}^{\mathsf{FHE}}(1^\lambda, p)$, *with* $q > 2\cdot p\cdot(B+1)\cdot 2^{\mathsf{sec}}$ *and* $\mathsf{sk}_1+\mathsf{sk}_2 = \mathsf{sk}$. *Let* (ct, m) *be a pair of ciphertext/plaintext messages and* m_1 *and* m_2 *values obtained with the 2-party distributed decryption procedure described above. Then, it holds that*

$$m = m_1 + m_2 \pmod p,$$

with probability at least $1 - N \cdot 2^{\mathsf{sec}}$.

Proof. First we notice that

$$m = ((d_1 + d_2) \pmod q) \pmod p,$$

and that we will always have $m = m_1 + m_2 \pmod p$ if the internal reduction modulo q in the decryption equation for m does not need to compensate for a wrap around. However, since we know ct is valid (i.e., $\mathsf{ct} \cdot (1, -\mathsf{sk}) = m + p \cdot \epsilon \pmod q$ with $\|\epsilon\|_\infty \leq B$) we also know that the coefficients of $d_1 + d_2 \pmod q$ will lie in the range $(-p\cdot(B+1), \ldots, p\cdot(B+1))$. Thus, the distributed decryption will potentially result in an error if and only if the coefficients of d_1 lie in one of the two ranges $(-q/2, -q/2+p\cdot(B+1))$ or $(q/2-p\cdot(B+1), q/2)$. Since each party added or subtracted the random R, it holds that d_1 is uniformly distributed in the range $(-q/2, \ldots, q/2)$. Therefore, the probability there is a wraparound in a single coefficient is bounded by $2 \cdot p \cdot (B + 1)/q < 2^{-\mathsf{sec}}$. However, we also known that, if there is a wrap around, it will definitely result in an invalid distributed decryption, as the error only consists of the addition of a single value of $q \pmod p \neq 0$. Thus, a single coefficient will be correct with probability $1 - 2^{-\mathsf{sec}}$. To obtain a correct decryption we need all coefficients to be correct, which will happen with probability

$$\left(1 - 2^{-\mathsf{sec}}\right)^N \approx 1 - N \cdot 2^{-\mathsf{sec}}.$$

□

We report details on the two party distributed decryption for the type msb in the full version.

5 Scooby: Multi-Party HEDS from LD-based FHE

In this section we detail how to construct a HEDS encryption scheme for the underlying ring R_p, from generic LD-based FHE. We call our construction Scooby, as it is similar to a spooky encryption but with a trusted setup. To denote the specific nature of this construction we refer to $\mathsf{SetUp}^{\mathsf{Scooby}}$, $\mathsf{Enc}^{\mathsf{Scooby}}_{\mathsf{pk}}$, etc., instead of $\mathsf{SetUp}^{\mathsf{HEDS}}$, $\mathsf{Enc}^{\mathsf{HEDS}}_{\mathsf{pk}}$, etc.

At the core of Scooby is the 2-party distributed decryption procedure described in the previous section. We show that, assuming an LD-based FHE scheme, this directly yields a 2-party Scooby. We then show how to bootstrap the 2-party scheme to the multi-party setting.

5.1 HEDS Key Generation

First, we need to slightly modify the KeyGen algorithm for the underlying FHE scheme to take a "special" form that is common to all standard FHE constructions. More concretely, the algorithm $\mathsf{KeyGen}^{\mathsf{FHE}}(1^\lambda, p)$ proceeds as follows, using two sub-procedures $\mathsf{ParamGen}()$ and $\mathsf{PubKeyGen}()$:

1. $\mathsf{params} \leftarrow \mathsf{ParamGen}(1^\lambda, p)$: This algorithm takes as input a security parameter λ, a plaintext modulo p and produces the scheme parameters $\mathsf{params} = (q, N, B, d, \Delta, S)$.
2. $\mathsf{sk} \leftarrow R_q^n$ such that $\|\mathsf{sk}\|_\infty \leq S$.
3. $\mathsf{pk} \leftarrow \mathsf{PubKeyGen}(1^\lambda, \mathsf{sk}, \mathsf{params})$: This algorithm, on input the secret key and scheme parameters, samples and outputs an associated public key pk.

5.2 Security Assumption

In our construction, we generate an FHE public key based on a secret-key $\mathsf{sk} = \mathsf{sk}_0 + \mathsf{sk}_1$, where $\mathsf{sk}_0, \mathsf{sk}_1$ are both sampled uniformly with coefficients bounded by the parameter S. For security, we require that the scheme defined by $(\mathsf{pk}, \mathsf{sk})$ satisfies the standard IND-CPA security notion, even when the adversary is given one of the original secret keys sk_i. This is formalized as follows.

Definition 5.1 (Bounded secret key IND-CPA security). *Let* $\mathsf{FHE} = (\mathsf{KeyGen}^{\mathsf{FHE}}, \mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}}, \mathsf{Dec}^{\mathsf{FHE}}_{\mathsf{sk}}, \mathsf{Eval}^{\mathsf{FHE}}_{\mathsf{pk}})$ *be a linear decryption-based FHE scheme, where* $\mathsf{KeyGen}^{\mathsf{FHE}}$ *is split into two sub-routines* $\mathsf{ParamGen}, \mathsf{PubKeyGen}$ *as above.*

We require that for $(q, N, B, d, \Delta, S) \leftarrow \mathsf{ParamGen}(1^\lambda, p)$*, and* $\mathsf{sk}_0, \mathsf{sk}_1 \leftarrow R_q^d$ *with* $\|\mathsf{sk}_i\| \leq S$*,* $\mathsf{sk} = \mathsf{sk}_0 + \mathsf{sk}_1$ *and* $\mathsf{pk} \leftarrow \mathsf{PubKeyGen}(\mathsf{sk})$*, it holds that for any PPT algorithm* $\mathcal{A}$*, for any* $\sigma \in \{0,1\}$*, messages* m_0, m_1 *and bit* $b \leftarrow \{0,1\}$*:*

$$\Pr[\mathcal{A}(1^\lambda, \mathsf{pk}, \mathsf{sk}_\sigma, \mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}}(m_b)) = b] \leq 1/2 + \mathsf{negl}(\lambda).$$

It is straightforward to verify that, given a linear decryption-based FHE scheme that satisfies the bounded secret-key IND-CPA security, we obtain a 2-party Scooby encryption scheme using the prior algorithms for 2-party distributed decryption into shares described in the previous section. Indeed, this

2-party distributed decryption forms the basis of the 2-party spooky construction in [18] and HSS construction in [9]. However, to obtain an n-party generalization is not immediate. A direct application of the trick used for 2-party to, say, 3-parties results in decryption errors due to unaccounted for wrap-arounds in the reduction modulo q of the local decryption. Coping with these wrap-arounds, without resorting to interaction, thus seems a challenge. A challenge which we solve in the next section.

5.3 From 2-party to n-party HEDS

Here we give the details of our construction Scooby, for n-party HEDS. The encryption and evaluation algorithms of Scooby are identical to that of the underlying linear decryption FHE scheme, so here we only describe the setup and share decryption procedures. We give two different variants of the construction, depending on whether the FHE scheme encodes the message in the lsb or msb of the ciphertext. In this section, we focus on a linear decryption FHE scheme that encodes the message in the lsb of the ciphertext; in the full version, we give a variant for the msb type.

Scooby Setup. Recall that the setup algorithm in HEDS takes as input a security parameter and outputs a global public key pk, as well as secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ to each of the n parties. For Scooby, in both the lsb and msb variants of LD-based FHE scheme, the underlying $\mathsf{SetUp}^{\mathsf{Scooby}}$ algorithm is the same. Note that in the following, the $\mathsf{SetUp}^{\mathsf{Scooby}}$ algorithm should be seen as a trusted setup procedure that is either run by a trusted third party, or executed via an MPC protocol, which can be done, for instance, based on the techniques from [28].

The $\mathsf{SetUp}^{\mathsf{Scooby}}$ algorithm is described in Figure 2. Recall that the main challenge is to setup up some key material which allows n parties to convert an FHE ciphertext into shares of the message, while using the 2-party distributed decryption method from the previous section. We build a binary tree with n leaves, where the original FHE ciphertext lives at the root node. We split the FHE secret key $\mathsf{sk}^{\mathsf{FHE}}$ into two shares $\widetilde{\mathsf{sk}}_0$, $\widetilde{\mathsf{sk}}_1$, and then generate a fresh FHE key pair for each of the two child nodes, and encrypt each $\widetilde{\mathsf{sk}}_b$, for $b \in \{0, 1\}$, under the corresponding public key. This process is repeated with the FHE secret keys generated for the children, and so on throughout the tree. Note that we abuse notation by writing $\mathsf{ct}_v = \mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}_v}(\widetilde{\mathsf{sk}}_v)$, even though $\widetilde{\mathsf{sk}}_v$ may not lie in the plaintext space; we implicitly assume here that $\widetilde{\mathsf{sk}}_v$ is broken up into bits (or possibly larger chunks), so ct_v is actually a vector of ciphertexts encrypting each bit separately.

The idea is that, during the decryption phase, the parties can homomorphically evaluate the 2-party distributed decryption function at each node of the tree, obtaining a share of the message, now encrypted under a child node's public key. The i-th party repeats this for each node on the path to leaf i, where it fi-

```
Algorithm $\mathsf{SetUp}^{\mathsf{Scooby}}(\lambda, p, n)$

The algorithm takes as input the security parameter $\lambda$, plaintext modulus $p$, and
number of parties $n$. It outputs a public key $\mathsf{pk}$ and secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.

1. Let $\mathsf{params} = (q, N, B, d, \Delta, S) \leftarrow \mathsf{ParamGen}(1^\lambda, p)$.
2. Sample a key $K^{\mathsf{prf}} \leftarrow \{0,1\}^\lambda$.
3. We construct a complete (but not necessarily full at the last layer) binary tree
   with $n$ leaves and height $h = \lceil \log(n) \rceil$, and index the levels from 0 up to $h$.
   Each node in level $i$ of the tree is labelled with a string of $i$ bits, so the root is
   the empty string $\perp$, and the children of node $v$ are $v\|0$ and $v\|1$.
4. Sample $\widetilde{\mathsf{sk}}_0, \widetilde{\mathsf{sk}}_1 \leftarrow R_q^d$ such that $\|\widetilde{\mathsf{sk}}_j\|_\infty \le S$.
5. Let $\mathsf{sk}_\perp^{\mathsf{FHE}} = \widetilde{\mathsf{sk}}_0 + \widetilde{\mathsf{sk}}_1$ and sample $\mathsf{pk}_\perp^{\mathsf{FHE}} = \mathsf{PubKeyGen}(1^\lambda, \mathsf{sk}_\perp^{\mathsf{FHE}}, \mathsf{params})$.
6. For each internal node $v$ (excluding the root and leaves) with children $v\|0$ and
   $v\|1$:
   (a) Sample $\widetilde{\mathsf{sk}}_{v\|0}, \widetilde{\mathsf{sk}}_{v\|1} \leftarrow R_q^d$ such that $\|\widetilde{\mathsf{sk}}_j\|_\infty \le S$.
   (b) Let $\mathsf{sk}_v^{\mathsf{FHE}} = \widetilde{\mathsf{sk}}_{v\|0} + \widetilde{\mathsf{sk}}_{v\|1}$, sample $\mathsf{pk}_v^{\mathsf{FHE}} = \mathsf{PubKeyGen}(1^\lambda, \mathsf{sk}_v^{\mathsf{FHE}}, \mathsf{params})$.
   (c) Let $\mathsf{ct}_v = \mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}_v}(\widetilde{\mathsf{sk}}_v)$.
7. Let $\mathsf{sk}_i$ contain the leaf secret key $\widetilde{\mathsf{sk}}_i$, together with $K^{\mathsf{prf}}$ and the public keys
   and ciphertexts on the path from the root to leaf $i$.
8. Output $\mathsf{pk} := \mathsf{pk}_\perp^{\mathsf{FHE}}$ and the secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
```

Fig. 2. Trusted setup algorithm for the Scooby construction

nally obtains a ciphertext encrypting an n-party sharing of the original message, which it can decrypt.

Given this setup procedure we define $\mathsf{Enc}^{\mathsf{Scooby}}_{\mathsf{pk}}$ and $\mathsf{Eval}^{\mathsf{Scooby}}_{\mathsf{pk}}$ exactly as is the case in the underlying LD-based FHE scheme. Next, we detail the $\mathsf{Dec}^{\mathsf{Scooby}}_{\mathsf{sk}_i}$ procedure in the lsb case.

Scooby Decryption. The decryption algorithms for Scooby in the lsb/msb-mode are described in Figure 3 and the full version, respectively. The decryption algorithm requires $\lceil \log n \rceil - 1$ evaluations of the $\mathsf{Eval}^{\mathsf{FHE}}$ function for the underlying LD-based FHE scheme, each for a different public key. Note that the circuit used in $\mathsf{Eval}^{\mathsf{FHE}}$ is almost exactly the decryption circuit, so the complexity of each of these homomorphic operations is the same as a bootstrapping operation in the underlying FHE scheme.

It is also clear that, due to the fact that at each internal branch we are homomorphically evaluating the two-party distributed decryption method from either Section 4.1 (for the lsb case) or the method for the msb case given in the full version, the final n messages m_i will sum up to the decryption of the ciphertext ct. The only difference is that instead of adding or subtracting a random nonce R, the parties are using the PRF F to randomize their shares in distributed decryption; thus, the correctness property of the scheme relies on the security of F.

```
Algorithm $\mathsf{Dec}^{\mathsf{Scooby}}_{\mathsf{sk}_i}(\mathsf{ct})$ (for lsb-based construction)

Let $F : \{0,1\}^\lambda \times [n] \to R_q$ be a pseudorandom function.

$\mathsf{Dec}^{\mathsf{Scooby}}_{\mathsf{sk}_i}(\mathsf{ct})$:
  1. Parse $\mathsf{sk}_i$ as $\widetilde{\mathsf{sk}}_i$, $K^{\mathsf{prf}}$ and $(\mathsf{pk}^{\mathsf{FHE}}_v, \mathsf{ct}_v)$, for every node $v$ from the root to leaf $i$.
  2. Let $\widetilde{\mathsf{ct}}_\perp := \mathsf{ct}$.
  3. For each internal node $v$ on the path from the root to leaf $i$ (excluding the root and leaf):
     (a) Write $v = u \| b$, where $u$ is the parent of $v$ (so $b = 0$ if $v$ is a left child and $b = 1$ otherwise).
     (b) Define the function:
         $$f^b_{\widetilde{\mathsf{ct}}_u} : \mathsf{sk} \mapsto \left(\widetilde{\mathsf{ct}}_u \cdot (b, -\mathsf{sk}) + (-1)^b \cdot F(K^{\mathsf{prf}}, u) \pmod q\right) \pmod p$$
     (c) Compute $\widetilde{\mathsf{ct}}_v := \mathsf{Eval}^{\mathsf{FHE}}_{\mathsf{pk}_v}(f^b_{\widetilde{\mathsf{ct}}_u}, \mathsf{ct}_v)$.
  4. Write $i = u \| b$, then take the leaf ciphertext $\widetilde{\mathsf{ct}}_i$ and output the share
         $$m_i = \left(\widetilde{\mathsf{ct}}_i \cdot (b, -\widetilde{\mathsf{sk}}_i) + (-1)^b \cdot F(K^{\mathsf{prf}}, u) \pmod q\right) \pmod p$$
```

Fig. 3. Decryption to shares for lsb-based Scooby

Theorem 5.1. *Let F be a pseudorandom function, and suppose there is an LD-like FHE scheme which satisfies the hardness assumption from Definition 5.1, such that $(q, N, B, d, \varDelta, S) \leftarrow \mathsf{ParamGen}(1^\lambda, p)$ with $q > 2 \cdot p \cdot (B+1) \cdot 2^{\mathsf{sec}}$. Then the* Scooby *construction in Fig. 2–3 is a secure n-party homomorphic encryption scheme with decryption to shares.*

The proof is given in the full version.

Remark 5.1. Note that for correctness to hold it is not sufficient that for a single party the path from the root to the node is correctly split. We need this to happen for *all* parties simultaneously. This means that the obtained probability is in fact $1 - n \cdot N \cdot 2^{-\sec}$ and not, as initially might be believed, $1 - \log(n) \cdot N \cdot 2^{-\sec}$.

A simpler variant relying on circular security. The previous construction avoids relying on a circular security assumption by switching to a freshly sampled FHE key at each node of the tree. We could instead simplify this slightly, with a variant of the construction where only one set of FHE secret keys is used. Here, we would start by sampling an independent secret key $\widetilde{\mathsf{sk}}_i$ for each leaf i. The public key associated with node v is then defined as $\mathsf{pk}_v = \mathsf{PubKeyGen}(1^\lambda, \mathsf{sk}_v, \mathsf{params})$, where sk_v is the sum of all the leaf secret keys that are descendants of v. We additionally encrypt sk under pk_v and give this out to the relevant parties. This introduces a circular security assumption, however, it does not seem to offer any significant efficiency benefits except for a slightly simpler setup algorithm.

5.4 BGV Parameters Supporting Scooby

It would appear that at first sight the parameters needed for Scooby are larger than those needed for standard FHE bootstrapping, due to the increase in q required by Equation (1). However, this is not necessarily the case, as we now explain in the case of the BGV encryption scheme.

Standard BGV decryption simply requires the bound $q > 2 \cdot p \cdot (B+1)$ for valid decryption, so we appear to have boosted the size of q by a factor of 2^{sec}. However, bootstrappable BGV as implemented in (say) HELib [24] utilizes an underlying levelled SHE scheme. At level zero, where no further homomorphic operations may take place without bootstrapping, we have a ciphertext modulus q_0 which satisfies $q_0 > 2 \cdot p \cdot (B+1)$. At level L, i.e., the initial encryption level, we have a ciphertext modulus q_L which satisfies $q_L > 2 \cdot p \cdot (B+1) \cdot 2^{b_p \cdot L}$, where b_p is the (average) bits-per-level of the chain of ciphertext moduli. On passing from each level from L down to zero, the size of the ciphertext modulus drops by (on average) 2^{b_p}. Note that, when bootstrapping a ciphertext from level zero, we do not end up with a ciphertext at level L, instead we obtain a ciphertext at level U (which denotes the so-called "usable" number of levels).

To see how this affects Scooby, we need to remember that at the end of the $\mathsf{Eval}_{\mathsf{pk}}^{\mathsf{Scooby}}$ procedure we will have a ciphertext at level U. This will satisfy our bound in Equation (1) if $2^{b_p \cdot U} \geq 2^{\mathsf{sec}}$. Then, in executing $\mathsf{Dec}_{\mathsf{sk}_i}^{\mathsf{Scooby}}$, at each level of the tree we notice that we are actually executing an operation equivalent to boostrapping. This is because at each node $v = u \| b$, where u is the parent node and $b \in \{0,1\}$, we are essentially either performing a homomorphic decryption with the key $(1, -\widetilde{\mathsf{sk}}_{u\|1}^{\mathsf{FHE}})$, or a homomorphic decryption with the key $(0, -\widetilde{\mathsf{sk}}_{u\|0}^{\mathsf{FHE}})$. Thus, at each stage of the execution of $\mathsf{Dec}_{\mathsf{sk}_i}^{\mathsf{Scooby}}$ we have a ciphertext ct which is at level U.

Examining the bootstrappable BGV parameters proposed in [24] we see that in all cases we have $2^{b_p \cdot U} \geq 2^{128}$. Thus the Equation (1) does not actually result in any increase in parameters, at least in the case of the BGV scheme.

6 Multi-Party HEDS from Weaker Assumptions

We now present alternative constructions to the previous section, without relying on FHE with linear decryption and a super-polynomial modulus. In the first construction, in Section 6.1, we use any generic FHE scheme and a 2-party HSS scheme that supports homomorphic evaluation of the FHE decryption circuit. This means we no longer need the local decryption trick from Section 4.1, so can use FHE based on LWE with a polynomial modulus [12]. All LWE-based FHE constructions have decryption in NC1, so the 2-party HSS can be instantiated based on the Paillier assumption [27] or on class groups [1], which support HSS for all of NC1.

In Section 6.2, we also give a variant of the construction that *only* requires 2-party HSS, and not FHE. This gives a way to bootstrap two-party HSS constructions to the multi-party setting. We show how it can be used to transform two-party HSS for branching programs, based on Paillier encryption, into 4-party HSS for homomorphic evaluation of constant-degree polynomials.

6.1 Scrappy: HEDS from Standard FHE + HSS for NC1

This construction, shown in Fig. 4, follows the tree-based structure of Scooby from the previous section. Previously, though, at each node of the tree, an FHE ciphertext was split into two ciphertexts encrypting shares of the message, by doing a special homomorphic decryption procedure tailored to the linear decryption property of the FHE scheme. In Scrappy, we instead do the homomorphic decryption procedure inside a 2-party HSS scheme. Since most FHE schemes have decryption in NC1, it suffices to rely on HSS for NC1, which can be built from non-LWE-based assumptions. Of course, if done naively, this means we no longer get encrypted shares of the previous message, but would actually obtain the shares directly due to use of HSS. To avoid leaking all intermediate shares, we use an additional FHE scheme on each level of the tree, and use this to homomorphically evaluate the HSS evaluation procedure. The HSS evaluation keys are then only given out at the leaves of the tree, while at higher levels they are encrypted under FHE. Note that we only need the weaker, private-key form HSS, from Remark 2.1, where the sharing algorithm can be seen as done by a trusted dealer.

Theorem 6.1. *Suppose there exists fully homomorphic encryption, and a 2-party HSS scheme that supports homomorphic evaluation of the FHE scheme's decryption circuit. Then, there exists an n-party homomorphic encryption scheme with decryption to shares, for any $n = \mathsf{poly}(\lambda)$.*

The proof is given in the full version.

Remark 6.1. The above theorem implies n-party HEDS assuming (1) LWE with a polynomial modulus [12], (2) circular security, and (3) HSS for NC1 circuits, which can be based on decisional composite residuosity [27] or a DDH-like assumption in class groups [1]. If we only require n-party HEDS for bounded-depth circuits, we can remove the circular security assumption, since we only required levelled FHE.

6.2 Shaggy: Bootstrapping HEDS to More Parties

We now give a separate transformation that increases the number of parties in HEDS, *without* relying on fully homomorphic encryption. The construction, in Fig. 5, essentially applies one layer of the previous, tree-based construction, with a branching factor of n instead of 2. Additionally, instead of alternating between FHE and HSS evaluation, we always evaluate within an n-party HEDS scheme. This allows bootstrapping any sufficiently powerful n-party HEDS to support n^2 parties.

$\mathsf{Scrappy}$: n-party HEDS from FHE + HSS

Let $(\mathsf{KeyGen}^{\mathsf{FHE}}, \mathsf{Enc}^{\mathsf{FHE}}, \mathsf{Eval}^{\mathsf{FHE}}, \mathsf{Dec}^{\mathsf{FHE}})$ be an FHE scheme and $(\mathsf{Share}^{\mathsf{HSS}}, \mathsf{Eval}^{\mathsf{HSS}})$ be a 2-party HSS scheme for the FHE decryption circuit.

– $\mathsf{SetUp}^{\mathsf{Scrappy}}(1^\lambda, n)$: Construct a complete binary tree of height $h = \lceil \log n \rceil$ and n leaves, and index the levels from 0 (at the root) up to h. Each node in level i of the tree is labelled with a string of i bits, so the root is labelled with the empty string $\perp$, and the children of node v are labelled $v\|0$ and $v\|1$.
  1. Sample a root key pair $(\mathsf{pk}_\perp, \mathsf{sk}_\perp) = \mathsf{KeyGen}^{\mathsf{FHE}}(1^\lambda)$.
  2. Sample HSS shares $(s_0, s_1) = \mathsf{Share}^{\mathsf{HSS}}(\mathsf{sk}_\perp)$.
  3. For each internal node v (excluding the root and leaf nodes), with parent node u and children $v\|0, v\|1$, compute the following values:
     (a) $(\mathsf{pk}_v, \mathsf{sk}_v) = \mathsf{KeyGen}^{\mathsf{FHE}}(1^\lambda)$.
     (b) $\mathsf{ct}^s_v = \mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}_v}(s_v)$
     (c) $(s_{v\|0}, s_{v\|1}) = \mathsf{Share}^{\mathsf{HSS}}(\mathsf{sk}_v)$.
  4. Output $\mathsf{pk} := \mathsf{pk}_\perp$ and the secret keys $\mathsf{sk}_i := \big(s_i, \{\mathsf{pk}_v, \mathsf{ct}^s_v\}_{v \in \mathsf{pathTo}(i)}\big)$, for $i = 0, \dots, n-1$.

– $\mathsf{Enc}^{\mathsf{Scrappy}}_{\mathsf{pk}}(m)$: Output $\mathsf{ct} = \mathsf{Enc}^{\mathsf{FHE}}_{\mathsf{pk}}(m)$.

– $\mathsf{Eval}^{\mathsf{Scrappy}}_{\mathsf{pk}}(C, (\mathsf{ct}_1, \dots, \mathsf{ct}_m))$: Output $\widetilde{\mathsf{ct}} = \mathsf{Eval}^{\mathsf{FHE}}_{\mathsf{pk}}(C, \mathsf{ct}_1, \dots, \mathsf{ct}_m)$.

– $\mathsf{Dec}^{\mathsf{Scrappy}}_{s_i}(\widetilde{\mathsf{ct}})$:
  1. Let $\widetilde{\mathsf{ct}}_\perp := \widetilde{\mathsf{ct}}$.
  2. For each internal node v on the path from the root to leaf i (excluding the root and leaf), with parent node u:
     (a) Define the pair of functions:
$$f_{\widetilde{\mathsf{ct}}_u} : \mathsf{sk} \mapsto \mathsf{Dec}^{\mathsf{FHE}}_{\mathsf{sk}}(\widetilde{\mathsf{ct}}_u)$$
$$g_{\widetilde{\mathsf{ct}}_u} : s_v \mapsto \mathsf{Eval}^{\mathsf{HSS}}(f_{\widetilde{\mathsf{ct}}_u}, s_v)$$
     (b) Compute $\widetilde{\mathsf{ct}}_v := \mathsf{Eval}^{\mathsf{FHE}}_{\mathsf{pk}_v}(g_{\widetilde{\mathsf{ct}}_u}, \mathsf{ct}^s_v)$.
  3. Output $y_i = \mathsf{Eval}^{\mathsf{HSS}}(f_{\mathsf{ct}}, s_i)$.

Fig. 4. Constructing n-party HEDS using standard FHE and 2-party HSS

Theorem 6.2. *Let n-*HEDS *be an n-party HEDS for a class of circuits $\mathcal{C}$, whose decryption algorithm, when viewed as a function of* sk_i*, can be written as a circuit in $\mathcal{C}$. Then, n^2-*HEDS *(in Figure 5) is an n^2-party HEDS for $\mathcal{C}$. Its encryption and evaluation algorithms are the same as in n-*Scooby*, while the complexity of decryption increases by a polynomial factor.*

The proof is given in the full version.

Note that the decryption complexity of the bootstrapped construction n^2-HEDS is increased by a polynomial factor. Depending on the original n-party scheme, then, it may not be possible to apply the transformation more than once, if the new decryption algorithm is no longer in the class $\mathcal{C}$.

Construction n^2-HEDS

Let n-HEDS be an n-party HEDS. We build n^2-party HEDS, and label the parties $P_{i,j}$, for $i,j \in [n]$

- $\mathsf{SetUp}^{\mathsf{n^2\text{-}HEDS}}(1^\lambda, n^2)$:
  1. Let $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) = \mathsf{SetUp}^{\mathsf{n\text{-}HEDS}}(1^\lambda, n)$.
  2. For $i \in [n]$:
     (a) Sample $(\mathsf{pk}_i, \mathsf{sk}_{i,1}, \ldots, \mathsf{sk}_{i,n}) = \mathsf{SetUp}^{\mathsf{n\text{-}HEDS}}(1^\lambda, n)$.
     (b) Sample $\mathsf{ct}_i^s = \mathsf{Enc}_{\mathsf{pk}_i}^{\mathsf{n\text{-}HEDS}}(\mathsf{sk}_i)$.
  3. Output pk and the n^2 secret keys $\mathsf{sk}_{i,j} := (\mathsf{ct}_i, \mathsf{sk}_{i,j}, \mathsf{pk}_i)$, for $i,j \in [n]$.

- $\mathsf{Enc}_{\mathsf{pk}}^{\mathsf{n^2\text{-}HEDS}}(m)$: Output $\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{n\text{-}HEDS}}(m)$.

- $\mathsf{Eval}_{\mathsf{pk}}^{\mathsf{n^2\text{-}HEDS}}(C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\rho))$:
  1. Compute $\mathsf{ct} = \mathsf{Eval}_{\mathsf{pk}}^{\mathsf{n\text{-}HEDS}}(C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\rho)$.
  2. Let f_{ct} be the function that takes as input sk_i and outputs $\mathsf{Dec}_{\mathsf{sk}_i}^{\mathsf{n\text{-}HEDS}}(\mathsf{ct})$.
  3. Compute $\widetilde{\mathsf{ct}}_i = \mathsf{Eval}_{\mathsf{pk}_i}^{\mathsf{n\text{-}HEDS}}(f_{\mathsf{ct}}, \mathsf{ct}_i^s)$, for $i = 1, \ldots, n$.
  4. Output $(\widetilde{\mathsf{ct}}_1, \ldots, \widetilde{\mathsf{ct}}_n)$.

- $\mathsf{Dec}_{\mathsf{sk}_{i,j}}^{\mathsf{n^2\text{-}HEDS}}(\widetilde{\mathsf{ct}}_i)$: Output $y_{i,j} = \mathsf{Dec}_{\mathsf{sk}_{i,j}}^{\mathsf{n\text{-}HEDS}}(\widetilde{\mathsf{ct}}_i)$.

Fig. 5. Bootstrapping n-party HEDS to n^2 parties

Instantiation with HSS from Paillier. We show how to apply the above transformation to two-party HSS based on the decisional composite residuosity assumption used in Paillier encryption. We can only apply the transformation once, so we obtain a 4-party HEDS/HSS scheme, which can support homomorphic evaluation of constant-degree polynomials. As the underlying two-party scheme n-HEDS, we can use the HSS construction from [27] or [29].

First, we need to frame the 2-party HSS constructions of [27, 29] in our HEDS framework. The constructions are given in a "public-key" flavour of HSS, with $\mathsf{SetUp}^{\mathsf{HSS}}$ and $\mathsf{Enc}^{\mathsf{HSS}}$ algorithms which are the same as in HEDS. The $\mathsf{Eval}^{\mathsf{HSS}}$ algorithm, however, requires knowing a secret key, unlike the syntax for $\mathsf{Eval}^{\mathsf{HEDS}}$. To make this fit our HEDS framework, we define $\mathsf{Eval}^{\mathsf{HEDS}}$ in the scheme to simply be the identity function, and move homomorphic evaluation into $\mathsf{Dec}_{\mathsf{sk}_i}^{\mathsf{HEDS}}$. This makes the resulting HEDS non-compact, but it can still be used for the construction in Fig. 5.

Complexity of Evaluation in Paillier-based HEDS. We now analyze the circuit complexity of the resulting $\mathsf{Dec}^{\mathsf{HEDS}}$ algorithm, which performs HSS evaluation of constant-degree polynomials. We can assume the polynomial is a simple monomial $f(x_1, \ldots, x_c) = x_1 x_2 \cdots x_c$ for a constant number of inputs (since to handle sums of monomials, it's enough to evaluate each monomial separately and add the shares).

With the methods of [27, 29], each input x_i is given as a Paillier encryption of x_i, together with encryptions of x_i multiplied with each bit of the secret key. In homomorphic evaluation, the parties perform $c-1$ sequential multiplications, where in each of these, the core operation is a step that computes:

$$z = \mathsf{DDLog}(C^d \mod N^2) + F_k(\mathsf{id}) \mod N$$

Here, $N = pq$ is a public modulus, $C \in \mathbb{Z}^*_{N^2}$ is a ciphertext, d is a secret share that is known only to one party, and F is a pseudorandom function with key k known to both parties. The distributed discrete log function $\mathsf{DDLog}(X)$ computes $\lfloor X/N \rfloor \cdot (X \bmod N)^{-1} \bmod N$.

In general, modular exponentiation and inversion are not known to be in NC1. However, it turns out that $\mathsf{DDLog}(C^d)$, when viewed as a function of d for fixed C, does lie in NC1. The idea is that since C is public, we can consider the powers $C^{2^j} \mod N^2$ as hard-coded into the description of the function. Similarly, we hardcode $C^{-2^j} \mod N$, for $j = 1, \ldots, \ell$, where ℓ is the bit length of d. This allows computing

$$C^d = \prod_{j=1}^{\ell} (C^{2^j})^{d_j} \bmod N^2, \quad (C^d \bmod N)^{-1} = \prod_{j=1}^{\ell} (C^{-2^j})^{d_j} \bmod N$$

Since iterated product, modular reduction, addition/subtraction and integer division are all in NC1 [2], $\mathsf{DDLog}(C^d)$ can be computed as an NC1 circuit. Furthermore, evaluation of a PRF based on factoring can be done in NC1 [26].

In the complete multiplication algorithm, the above step is repeated $O(\lambda)$ times in parallel, which does not affect the circuit depth. The multiplication algorithm is run c times sequentially, where the outputs of one multiplication are used as the private d shares input to the next. If c is a constant, it follows that the entire evaluation procedure is in NC1.

Plugging in two-party HSS for poly-sized branching programs (which includes NC1), we obtain the following.

Corollary 6.1. *Assume the decisional composite residuosity assumption holds. Then, there exists a 4-party (non-compact) homomorphic encryption scheme with decryption to shares for constant-degree polynomials.*

Acknowledgements

The work of authors from KU Leuven was supported by CyberSecurity Research Flanders with reference number VR20192203 and by the FWO under an Odysseus project GOH9718N. Peter Scholl was supported by the Independent Research Fund Denmark (DFF) under project number 0165-00107B (C3PO) and the Aarhus University Research Foundation (AUFF).

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Cyber Security Research Flanders, the FWO, DFF or AUFF.

References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. Cryptology ePrint Archive, Report 2022/363 (2022), https://ia.cr/2022/363
2. Beame, P., Cook, S., Hoover, H.: Log depth circuits for division and related problems. In: 25th Annual Symposium onFoundations of Computer Science, 1984. pp. 1–6 (1984)
3. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 489–518. Springer, Heidelberg (Aug 2019)
4. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Correlated pseudorandom functions from variable-density LPN. In: 61st FOCS. pp. 1069–1080. IEEE Computer Society Press (Nov 2020)
5. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (Apr 2015)
6. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (Aug 2016)
7. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1292–1303. ACM Press (Oct 2016)
8. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Karlin, A.R. (ed.) ITCS 2018. vol. 94, pp. 21:1–21:21. LIPIcs (Jan 2018)
9. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 3–33. Springer, Heidelberg (May 2019)
10. Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 407–437. Springer, Heidelberg (Dec 2019)
11. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012)
12. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) ITCS 2014. pp. 1–12. ACM (Jan 2014)
13. Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from TFHE. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 446–472. Springer, Heidelberg (Dec 2019)
14. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 395–412. ACM Press (Nov 2019)
15. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (Dec 2016)

16. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology 33(1), 34–91 (Jan 2020)
17. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled FHE from learning with errors. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 630–656. Springer, Heidelberg (Aug 2015)
18. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (Aug 2016)
19. Doerner, J., shelat, a.: Scaling ORAM for secure computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 523–535. ACM Press (Oct / Nov 2017)
20. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144
21. Gentry, C., Halevi, S.: Compressible FHE with applications to PIR. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 438–464. Springer, Heidelberg (Dec 2019)
22. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (Aug 2013)
23. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (May 2014)
24. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology 34(1), 7 (Jan 2021)
25. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (May 2016)
26. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. Journal of the ACM 51(2), 231–262 (2004)
27. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of paillier: Homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 678–708. Springer, Heidelberg (Oct 2021)
28. Rotaru, D., Smart, N.P., Tanguy, T., Vercauteren, F., Wood, T.: Actively secure setup for SPDZ. Journal of Cryptology 35(1), 5 (Jan 2022)
29. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 687–717. Springer, Heidelberg, Virtual Event (Aug 2021)