Multi-Key Homomorphic Secret Sharing

Geoffroy Couteau[1], Lalita Devadas[2], Aditya Hegde[3]*, Abhishek Jain[3,4], and Sacha Servan-Schreiber[2]**

[1] CNRS, IRIF, Université Paris Cité
[2] MIT
[3] JHU
[4] NTT Research

Abstract. Homomorphic secret sharing (HSS) is a distributed analogue of fully homomorphic encryption (FHE) where following an input-sharing phase, two or more parties can locally compute a function over their private inputs to obtain shares of the function output.

Over the last decade, HSS schemes have been constructed from an array of different assumptions. However, all existing HSS schemes, except ones based on assumptions known to imply multi-key FHE, require a public-key infrastructure (PKI) or a correlated setup between parties. This limitation carries over to many applications of HSS.

In this work, we construct *multi-key* homomorphic secret sharing (MKHSS), where given only a common reference string (CRS), two parties can secret share their inputs to each other and then perform local computations as in HSS, eliminating the need for PKI or a correlated setup. Specifically, we present the first MKHSS schemes supporting all NC^1 computations from either the decisional Diffie–Hellman (DDH) assumption, the decisional composite residuosity (DCR) assumption, or DDH-like assumptions in class group.

Our constructions imply the following applications in the CRS model:

- **Succinct two-round secure computation.** Under the same assumptions as our MKHSS schemes, we construct a succinct, two-round, two-party secure computation protocol for NC^1 circuits. Previously, such a result was only known from the learning with errors assumption.
- **Attribute-based NIKE.** Under DCR or class group assumptions, we construct non-interactive key exchange (NIKE) protocols where two parties agree on a key if and only if their secret attributes satisfy a public NC^1 predicate. This significantly generalizes the existing notion of password-based NIKE.
- **Public-key PCFs.** Under DCR or class group assumptions, we construct public-key pseudorandom correlation functions (PCFs) for any NC^1 correlation. This yields the first public-key PCFs for Beaver triples (and more) from non-lattice assumptions.
- **Silent MPC.** Under DCR or class group assumptions, we construct a p-party secure computation protocol in the silent preprocessing model where the preprocessing phase has communication $O(p)$, ignoring polynomial factors. All prior protocols that do not rely on multi-key FHE techniques require $\Omega(p^2)$ communication.

* This work was done in part while the author was at IRIF.
** This work was done in part while the author was at NTT Research.

Table of Contents

1 Introduction

In a homomorphic secret sharing (HSS) [BGI16] scheme supporting a class of functions $\mathcal{F}$, two or more parties, each with a share of a secret x, can compute a function $f \in \mathcal{F}$ over x to get a share of the result $f(x)$. For security, any strict subset of the shares must hide the secret x. Importantly, the shares must be homomorphic, in that they must support local (non-interactive) function evaluations such that for any $f \in \mathcal{F}$, the output shares correspond to an additive sharing of $f(x)$.

The key property of HSS is *succinctness*, namely, the size of the input and output shares must be independent of the size of the circuit computing the function over the secret-shared inputs. HSS can be viewed as a distributed analogue of fully homomorphic encryption [Gen09]: it allows two parties to securely compute a function over their private inputs succinctly with respect to the circuit size.

Starting from the seminal work of Boyle, Gilboa, and Ishai [BGI16], a key goal of HSS is to build secure computation schemes from assumptions other than learning with errors (LWE). By now, many HSS schemes are known [BGI16, DHRW16, BGI17, BCG$^+$17, BKS19, BCG$^+$19b, CM21, OSY21, RS21, ADOS22, DIJL23] from a variety of standard assumptions that vary in the number of parties (most schemes support two parties), the class of supported functions (most schemes support NC^1 computations), and correctness error. While some HSS schemes have a non-negligible correctness error, they still have applications to secure computation [BGI17, DIJL23].

Over the last decade, HSS schemes have enabled a variety of applications in cryptography. These applications include secure computation with sublinear communication [BGI16, Cou19, CM21, DIJL23], private information retrieval [GI14, BGI16], pseudorandom correlation generators [BCGI18, BCG$^+$19b], constrained pseudorandom functions [CMPR23], and more.

	Assumption	CRS	Transparent Setup	Computational Class	Error
[DHRW16]	$i\mathcal{O}$+DDH	✗	✓	$\mathsf{P/poly}$	negl
[DHRW16, XW23]	LWE	✓	✓	$\mathsf{P/poly}^\star$	negl
Theorem 1	NIDLS DDH	✓	✗†	NC^1	negl
Theorem 2	DDH	✓	✓	NC^1	$1/\mathsf{poly}$
Theorem 6	DCR	✓	✗	NC^1	negl

Table 1: Constructions of MKHSS realized in this work and comparison to prior work.
*Requires making circular security assumptions to obtain a scheme for all circuits.
†When instantiated with class groups, the setup is transparent.

Homomorphic secret sharing with multiple inputs. The basic notion of HSS only enables computations over the private input of a single party. To support multiple private inputs, the notion of public-key HSS [BGI17, ADOS22] was proposed. In a public-key HSS scheme, following a CRS setup, there is a public-key setup phase where the parties sample and publish their respective public keys. Using these public keys, the parties locally derive a common public key and use it for secret sharing their inputs with one another.[5] This enables the parties to perform secure computations over their private inputs encrypted under the common public key.

Intuitively, public-key HSS can be viewed as an analogue of *threshold* fully homomorphic encryption [AJL+12]—a multi-party version of FHE in the public-key infrastructure (PKI) model. The key drawback of this notion is the necessity of the PKI setup, which itself relies on a CRS, prior to the input sharing phase. This limitation, in turn, affects the applications of HSS. For example, the PKI requirement carries over in the application of HSS to sublinear secure computation, imposing a minimum of three rounds, which is suboptimal [HLP11].

Furthermore, while HSS (with negligible correctness error) for NC^1 computations implies pseudorandom correlation functions (PCFs) for NC^1 correlations [BGMM20, CMPR23] (assuming the existence of pseudorandom functions computable in NC^1), the same is not known for *public-key* PCFs. A public-key PCF [OSY21, BCM+24] is a much stronger primitive that allows two parties to generate correlated randomness "on the fly," without needing to engage in a correlated setup ahead of time.

Multi-key homomorphic secret sharing. In this work, we study *multi-key* homomorphic secret sharing (MKHSS), which does not require any PKI or other correlated-randomness setup. Instead, given only a CRS, the parties can directly secret share their inputs to each other. In this sense, MKHSS can be viewed as an analogue of multi-key FHE [LTV12, MW16]—a multi-party version of FHE that enables computing over the private data of multiple entities, without needing PKI.

MKHSS for multiple parties is readily implied by spooky encryption [DHRW16], which is currently only known from assumptions known to imply FHE. Similarly to the foundational work of Boyle et al. [BGI16], which initiated the study of HSS as an alternative to FHE, we investigate the feasibility of MKHSS from assumptions not known to imply FHE. As we discuss next, MKHSS enables new applications that were not previously known from public-key HSS, similarly to how multi-key FHE enabled many new applications relative to the older (and weaker) notion of threshold FHE.

We show that *multi-key* HSS is possible from group-based assumptions in the two-party setting and prove the following theorem:

Informal Theorem 1 (Existence of MKHSS). *There exist the following instantiations of a two-party, multi-key homomorphic secret sharing scheme for computing all functions in the class* NC^1*:*

(1) Under the DDH assumption over cyclic groups, with a transparent setup, and inverse polynomial correctness error.
(2) Under the DDH assumption and small exponent assumption over class groups, with transparent setup, and negligible correctness error.
(3) Under the DCR assumption, with a trusted setup, and negligible correctness error.

[5] Alternatively, a correlated setup can take place where the parties obtain shares of a secret key belonging to a common public key.

In fact, the construction over class groups is an instantiation of a general template for MKHSS schemes in the non-interactive discrete log sharing (NIDLS) framework of Abram et al. [ADOS22]. The same template yields MKHSS schemes under the DDH and small exponent assumptions over the group used for Paillier encryption, as well as an extension of the Joye–Libert cryptosystem described by Abram et al. [ADOS22].

Similar to prior works, our MKHSS schemes support evaluating polynomial size restricted multiplication straight-line (RMS) programs—arithmetic circuits over the integers with restrictions on the inputs to multiplication gates, which contain the class NC^1.

Our DCR and class group instantiations have a negligible correctness error and support an exponentially large message space, similar to non-multi-key HSS constructions from these assumptions [RS21, OSY21, ADOS22]. In addition, the class group instantiation offers a transparent setup, which is not the case for the DCR construction where the CRS is structured. Our DDH instantiation works over any Diffie–Hellman group and has the benefit of having a transparent setup. However, similar to non-multi-key HSS constructions from DDH [BGI16], it only supports a polynomial size message space and has an inherent (but tunable) inverse polynomial correctness error.

We summarize our results in Table 1.

1.1 Applications of multi-key HSS

We show that many of the applications that multi-key FHE implies are also possible from MKHSS in the two-party setting, thanks to our constructions. We briefly summarize the applications of our schemes. Technical details surrounding these applications can be found in Sections 5 and 6.

Sublinear, two-round secure computation. As was shown by Boyle et al. [BGI17], standard HSS already gives two-party secure computation in three rounds and with sublinear communication in the circuit size. At a high level, the three-round protocol proceeds as follows:

Round 1: Agree on a public key and derive shares of the secret key.

Round 2: Exchange inputs encrypted under the public key.

Round 3: Locally evaluate the function and output the resulting shares.

The question of constructing *two-round* sublinear secure computation protocols from group-based assumptions has remained open. In particular, only *multi-key* FHE was known to be sufficient to instantiate sublinear two-round secure computation in the CRS model.

Our constructions of MKHSS give the first realization of sublinear, two-round, two-party secure computation in the CRS model from assumptions that are not known to imply FHE. Concretely, multi-key HSS immediately implies the following sublinear, two-round secure computation protocol:

Round 1: Exchange inputs encrypted under independent public keys.

Round 2: Locally evaluate the function and output the resulting shares.

Importantly, this protocol achieves *reusability* of the first round messages [BL20, BGMM20, AJJM20, BJKL21, AJJM21] in the following sense: the parties can compute different functions over their inputs without having to recompute their first round messages. Furthermore, a party can reuse its first-round message in different computations with different parties. In particular, this enables one party to even go offline after sending the first message, and only later complete the computation asynchronously.

Informal Theorem 2. *Under each of the three instantiations of multi-key homomorphic secret sharing from Informal Theorem 1, there exists a sublinear, two-party, two-round secure computation protocol for computations in the class* NC^1.

Attribute-based NIKE. We show that our constructions of MKHSS also imply *attribute-based non-interactive key exchange* (ANIKE) supporting NC^1 predicates.

An ANIKE scheme involves two parties, each with their own secret attribute. The requirement is that the parties can derive a shared key if their secret attributes jointly satisfy a public predicate. However, if their attributes do not satisfy the predicate, then they derive independent keys (and do not learn that the predicate was unsatisfied).

ANIKE captures password-based NIKE as well as its extensions such as *fuzzy* password-based NIKE, where parties can derive a shared key only if they share approximately-matching passwords (e.g., those derived from biometrics). In general, ANIKE is well-suited for applications that involve authenticating complicated credentials before providing sensitive information.

We briefly summarize the construction (details are provided in Section 5.2).

Public key: Alice samples an MKHSS public and secret key, and generates input shares of her secret attribute x_A and a random shift. Her public key consists of her MKHSS public key and the input share of her attribute and shift. Bob computes his public key in a symmetric manner.

Key derivation Alice, given (1) her secret key, (2) her own input share, and (3) Bob's public key with an input share of his secret attribute x_B, uses MKHSS to evaluate the program which computes the predicate and multiplies the result by the random shift of each party. Bob evaluates his shares exactly as Alice does. The key for each party consists of the subtractive share output by the MKHSS evaluation.

If the predicate evaluates to 0, which we define as the predicate being satisfied, Alice and Bob end up with pseudorandom subtractive shares of 0, i.e., the same key. On the other hand, if the predicate evaluates to 1, Alice and Bob end up with subtractive shares of a random value (i.e., independent keys) because of the random shifts.

To the best of our knowledge, all existing constructions of key exchange which support NC^1 predicates require interaction and/or assume some idealized model [KKL$^+$16, Mel22]. We realize the first *non-interactive* construction in the standard model. Thanks to the non-interactivity property, our ANIKE scheme—and the associated security proof—is conceptually very simple. In contrast, interactive constructions of ANIKE have many moving parts, have complicated proofs as a result, and many constructions have been broken due to subtle flaws [JRX24].

Informal Theorem 3. *Under the DCR assumption or the DDH and small exponent assumption in class groups, there exists an attribute-based non-interactive key exchange protocol supporting predicates in the class* NC^1.

Public-key PCFs for NC^1 correlations. Modern secure computation protocols are realized in the preprocessing model [Bea95, DPSZ12]. In this model, during an "offline" preprocessing phase, the computing parties generate a large amount of *pseudorandom correlations* that are independent of any function they will later compute. Then, during an online phase, the parties use the stored correlations to compute a function over their inputs in a secure protocol. Thanks to the correlated randomness, the online phase has far greater efficiency by not requiring any cryptographic operations. Pseudorandom correlation functions [BCG$^+$20a] (PCFs) push this model of secure computation to the limit by allowing parties to obtain a short key that they can use to locally, and "on-demand," to generate correlated pseudorandomness for use in the online phase.

Starting with the work of Orlandi, Scholl, and Yakoubov [OSY21], which introduced the concept of a public-key PCF, parties can *non-interactively* derive a PCF key using only the other party's public key. The advantage of public-key PCFs is that any pair of parties can generate correlated randomness on the fly, using only each other's public keys. However, existing constructions of public-key PCFs [OSY21, BCM$^+$24, CDD$^+$24], are restricted to the OT/VOLE correlation (or weaker variants thereof). In particular, barring spooky encryption, constructing a public-key PCF for even Beaver triple correlations has, so far, remained elusive.

In Section 6, we use our MKHSS constructions to build the first public-key PCF for any NC^1 correlation, from either the DCR assumption or the DDH and small exponent assumption in class groups.

Informal Theorem 4. *Under the DCR assumption or the DDH and small exponent assumption in class groups, there exists a public-key pseudorandom correlation function for* NC^1 *correlations.*

Remark 1 (Public-key PCFs with a transparent setup). We remark that our approach to constructing public-key PCFs from MKHSS is markedly different from prior constructions of public-key PCFs. Specifically, prior work on public-key PCFs [OSY21, ADOS22] exploit properties of HSS schemes built from the Paillier or Goldwasser–Micali encryption scheme to realize the public-key setup (e.g., the fact that these encryption schemes have have a dense ciphertext space). This prevents the existing approaches to public-key PCFs to have a transparent setup. In contrast, our approach via MKHSS is generic, which allows us to obtain a public-key PCF with a transparent setup (from class groups). Such a result was not known before, even for the OT/VOLE correlation.

Silent, secure multi-party computation. For multi-party computation protocols instantiated in the preprocessing model, the typical choice of correlated randomness consists of Beaver triples [Bea92]. A p-party Beaver triple consists of additive shares of (a, b, ab), where a, b are random elements in some finite ring. In practice, the cost of securely generating the correlated randomness in the preprocessing phase often dominates the overall cost of the protocol. To mitigate this cost, a relatively recent line of work [BCGI18, BCG$^+$19a, BCG$^+$19b, BCG$^+$20a] has introduced the *silent preprocessing model*, in which the correlated randomness is replaced by correlated *pseudo*randomness computed by a PCF.

Thanks to recent advances in homomorphic secret sharing and PCFs, there exist silent preprocessing protocols for generating suitable correlated pseudorandomness from standard assumption such as DCR, DDH and the small exponent assumption in class groups, and variants of LPN [BCG$^+$20b, OSY21, RS21, ADOS22, BCCD23]. However, in the context of p-party secure computation, all known methods (that do not rely on spooky encryption) incur an $\Omega(p^2) \cdot \mathsf{poly}(\lambda)$ communication overhead in the preprocessing phase. This overhead stems from the fact that all known constructions of HSS and PCFs are, barring some exceptions (cf. Remark 2), restricted to the setting of *two participants*, and generating p-party correlations via these primitives requires all pairs of parties to interact.

Remark 2. Nearly all HSS and PCF constructions are restricted to the two-party setting. However, some exceptions include the p-party HSS scheme from sparse LPN of Dao, Ishai, Jain, and Lin [DIJL23], which cannot be used to generate correlated randomness due to its imperfect correctness, the 4-party DCR-based HSS scheme of Boyle, Couteau, and Meyer [BCM23], and the 8-party scheme of Couteau and Kumar [CK24]. We also note that this restriction also applies to pseudorandom correlation generators (PCGs) [BCG$^+$19b].

Using MKHSS, we show how to construct a multi-party, public-key PCF for Beaver triples. At a high level, using our multi-party public-key PCF for NC^1, p parties can simultaneously broadcast their public keys on a public channel. Then, any pair of parties can, without any interaction, derive two-party Beaver triples. Using the two-party shares, all parties can locally aggregate their two-party Beaver triples to obtain (an arbitrary number of) p-party Beaver triples. Using these precomputed (and pseudorandom) Beaver triples, the parties can then run any efficient p-party non-cryptographic protocol to securely compute a target function (e.g., via the GMW protocol [GMW87]).

As a direct corollary of our MKHSS constructions, under DCR or DDH and small exponent assumption over class group, there exists a p-party protocol securely computing an arithmetic circuit C with s multiplication gates and m outputs over a ring $\mathcal{R}$ with the following communication complexity:

– In the preprocessing phase, the parties communicate $p \cdot \mathsf{poly}(\lambda)$ bits in a single broadcast round.
– In the online phase, the parties communicate $p \cdot (2s + m)$ elements of $\mathcal{R}$.

This yields a quadratic communication improvement over the state-of-the-art [BBC$^+$24] approach for secure computation in the preprocessing phase. However, we note that our constructions are still primarily of theoretical interest because our MKHSS constructions are not concretely efficient for general computations.

Informal Theorem 5. *Let C be an arithmetic circuit with n inputs, s multiplication gates, and m outputs, instantiated over a ring $\mathcal{R}$. Under the DCR assumption or the DDH and short exponent assumption in class groups, for any number of parties p, there exists a p-party secure computation protocol for computing C in the preprocessing model, with the following communication complexity:*

– *In the preprocessing phase: $O(p)$ bits in a single broadcast round.*
– *In the online phase: $p \cdot (2s + m)$ ring elements.*

The protocol is secure against a passive adversary corrupting any strict subset of parties.

Paper organization. We provide an in-depth technical overview in Section 2 capturing the details of our constructions. In Section 3, we provide the necessary preliminaries related to our constructions. In Section 4, we provide our formal definition of MKHSS. In Section 4.3, we describe our constructions of MKHSS under the DDH and small-exponent assumption in the Paillier group or class groups. In Section 4.4, we describe our construction of MKHSS from DDH. In Sections 5.2 and 6, we provide detailed constructions of these applications.

Supplementary material. In Appendix A.4, we provide an alternative construction of MKHSS solely from the DCR assumption using a different approach. Specifically, this alternative scheme avoids having to make the DDH and small exponent assumptions in the Paillier group.

2 Technical Overview

In this section, we provide a technical overview of our constructions. We organize the overview into the following subsections:

– *Background.* In Section 2.1, we define the notation that we use. Then, in Section 2.2, we describe the basic template underpinning all existing group-based HSS constructions and provide other relevant background.

– *Challenges.* In Section 2.3, we explain the challenges involved in adapting the basic HSS template into a multi-key HSS construction.

– *Construction in the NIDLS framework.* In Sections 2.4 and 2.5, we explain the new ideas that allow us to build MKHSS constructions from DCR and DDH over class groups using the NIDLS framework of Abram et al. [ADOS22].

Construction from DDH. In Section 2.6, we describe the challenges involved in adapting the ideas from the NIDLS framework to the DDH setting. In Section 2.7, we describe how we resolve these challenges to realize MKHSS from DDH.

2.1 Notation

We briefly provide some relevant notation for this overview, see Section 3 for more details. We let λ denote the security parameter. We let $a \leftarrow \mathsf{Alg}$ denote the output of a (possibly randomized) algorithm Alg and $a \leftarrow\!\!\$\, S$ denote a uniformly random sampling from the set S. Assignment of a value b to a variable a is denoted $a := b$. We denote three types of "secret shares" which form the backbone of existing HSS scheme abstractions [BGI16] (see Section 2.2 for background) as follows:

– *Input shares* of a message x are denoted by $[\![x]\!]$.
– *Memory shares* of a message x are denoted by $\langle\!\langle x \rangle\!\rangle$.
– *Subtractive shares* of a message x are denoted by $\langle x \rangle$.[6]

This notation is used to describe the *set* of shares of the message x. When referring to a single party's share, we write $[\![x]\!]_\sigma$, $\langle\!\langle x \rangle\!\rangle_\sigma$, and $\langle x \rangle_\sigma$, where $\sigma \in \{A, B\}$ is the party identifier (e.g., Alice and Bob's subtractive shares of x are denoted $\langle x \rangle_A$ and $\langle x \rangle_B$, respectively). We define these share types and describe how they are used to realize an HSS scheme next.

2.2 Background on HSS from group-based assumptions

Here, we describe a **simplified** template capturing the basics of existing group-based HSS constructions [BGI16, BCG+17, OSY21, RS21, ADOS22]. Relative to the full constructions, this minimal template omits some important details in the interest of clarity. Our primary goal here is to capture the essential components needed to understand our multi-key HSS approach. Because all known group-based HSS schemes are in the two-party setting, we will call these parties Alice and Bob throughout this overview.

Instantiating the group. Group-based HSS constructions require an Abelian group $\mathbb{G}$ in which a suitable subgroup indistinguishability assumption holds (e.g., DDH in cyclic groups, DCR in the Paillier group, or similar assumptions in class groups). We will use g to denote the generator of $\mathbb{G}$. We will also use a "special" generator h for a suitable subgroup of $\mathbb{G}$ in which the discrete logarithm is computationally easy.[7] Later, we will instantiate $\mathbb{G}$ from several assumptions using the NIDLS framework [ADOS22] and describe a separate construction from DDH.

Correlated setup. All existing HSS schemes require some form of correlated setup process to generate a common public key and distribute "evaluation keys" to the two parties. More concretely, in group-based constructions, the setup produces a public key $f := g^{-s}$ and secret shares the secret key s between Alice and Bob. Following the trusted setup, each party can generate *input shares*

[6] Subtractive shares (z_A, z_B) of a message x are defined over the integers such that $x = z_A - z_B$.
[7] The Paillier [Pai99] group $\mathbb{G} = \mathbb{Z}^*_{N^2}$ is an example of where there exist such a g and h under DCR.

of a private input x by encrypting it under the public key f with an "ElGamal-style" encryption over $\mathbb{G}$. Looking ahead, the main challenge in realizing *multi-key* HSS is replacing the entire trusted setup process with a common reference string (or even just a common *random* string). In particular, while it was shown that it is possible to reduce the correlated setup down to one round of interaction [BGI17, ADOS22] in the PKI model (i.e., when all parties know each other's public keys), removing this round of interaction has remained an open problem.

Input shares. An input share of a message x (we will define the message space later) under the public key f consists of two "ElGamal-like" ciphertexts in the group, where the first ciphertext encrypts $x \cdot s$ (recall, s is the secret key) and the second ciphertext encrypts x. All operations over the messages are performed "in the exponent" of the subgroup of $\mathbb{G}$ generated by h. An HSS input share of a message x given to party $\sigma \in \{A, B\}$ is denoted as $[\![x]\!]_\sigma$ and defined as:

$$[\![x]\!]_\sigma := \Big(\underbrace{(g^r,\ h^{x \cdot s} f^r)}_{\text{Ciphertext 1}},\ \underbrace{(g^{r'},\ h^x f^{r'})}_{\text{Ciphertext 2}} \Big), \tag{1}$$

where $r, r' \leftarrow\$ \mathbb{Z}_N$. Note that all parties get the same ciphertexts; while it is possible to add private state to the input shares, group-based HSS schemes satisfy the property that at least one component of the input share is identical across parties.

In addition to input shares, existing HSS schemes define an "intermediate" sharing used during a computation called a *memory share*, which we describe next.

Memory shares. A memory share of a message x held by party $\sigma \in \{A, B\}$ is denoted as $\langle\!\langle x \rangle\!\rangle_\sigma$ and is defined as a tuple of secret shares, consisting of subtractive shares of the message x and $x \cdot s$. In particular, a memory share is the secret-shared analog of an input share. That is,

$$\langle\!\langle x \rangle\!\rangle_\sigma := \Big(\langle x \cdot s \rangle_\sigma, \langle x \rangle_\sigma \Big). \tag{2}$$

Using the definition of an input share and a memory share, we can now describe how existing group-based HSS schemes evaluate functions.

2.2.1 Evaluating functions

The template for evaluating functions on the input shares, introduced by Boyle et al. [BGI16], is to emulate the program via a set of multiplication and addition instructions. In particular, the idea is to show that it is possible to compute a restricted-multiplication straight-line (RMS) program [Cle90]—a special model of computation that is known to be sufficiently powerful to evaluate all branching programs (and the class of functions in NC^1).[8]

In a nutshell, RMS programs are defined to take a set of input values and require maintaining the following rules. Each input to the program can either be (1) converted into a memory value or (2) multiplied by a memory value to produce a memory value of the product [Cle90, BGI16]. Moreover, (3) memory values can be added together to produce a memory value of the sum. In particular, what an RMS program does *not* allow is multiplying two memory values together, since that would imply the ability to compute all functions.

In existing group-based HSS schemes, non-interactively evaluating RMS programs over input shares boils down to computing (2)—a multiplication between an input share and a memory share. That is, given an input share of x and a memory share of y, it should be possible for each party to *locally* derive a memory share of xy. Once this single requirement is satisfied, meeting the other requirements becomes relatively straightforward.

The fact that the input and memory shares defined in Equations (1) and (2) enable computing a memory share of the product is not difficult to show, but requires using one crucial ingredient: the distributed discrete logarithm (DDLog) procedure [BGI16, OSY21, RS21, ADOS22].

Tool: The Distributed Discrete Logarithm. The DDLog procedure enables local conversion of multiplicative shares to subtractive shares as follows. Given *multiplicative* shares of any value x in the group $\mathbb{G}$, where one party holds $h^{\langle x \rangle_A} g^{\langle 0 \rangle_A}$ and the other party holds $h^{\langle x \rangle_B} g^{\langle 0 \rangle_B}$ such that[9]

$$h^{\langle x \rangle_A} g^{\langle 0 \rangle_A} \ \cdot\ h^{-\langle x \rangle_B} g^{-\langle 0 \rangle_B} = h^x,$$

[8] See also Section 3 for background on RMS programs.
[9] Note that h is the group element of order N in $\mathbb{Z}^*_{N^2}$.

the DDLog procedure allows party-σ to obtain a subtractive share $\langle x \rangle_\sigma$. The details of the distributed discrete log procedure do not matter for the purposes of this overview, and we will treat it as a black-box algorithm satisfying the above "share conversion" property. However, it does play a vital role in computing multiplications between input shares and memory shares in all existing group-based HSS schemes, as we explain next.

Computing a multiplication in HSS. Computing a multiplication between an input share and a memory share is done in two steps. The idea is to exploit (1) the additive homomorphism of memory shares, (2) the additive homomorphism of the ElGamal-style encryption, and (3) the linear decryption process. First, the parties compute the multiplication "in the exponent" of the group. Then, using DDLog, the resulting multiplicative shares are converted back to memory shares. In more detail:

Step I: Computing a multiplication "in the exponent." Given an input share $[\![x]\!]_\sigma$ and a memory share $\langle\!\langle y \rangle\!\rangle_\sigma$, party-$\sigma$ (for $\sigma \in \{A, B\}$) computes:

$$\Big((g^r)^{\langle y \cdot s \rangle_\sigma} \cdot (h^{x \cdot s} f^r)^{\langle y \rangle_\sigma},\ (g^{r'})^{\langle y \cdot s \rangle_\sigma} \cdot (h^x f^{r'})^{\langle y \rangle_\sigma}\Big) = \Big(h^{\langle xy \cdot s \rangle_\sigma} g^{\langle 0 \rangle_\sigma},\ h^{\langle xy \rangle_\sigma} g^{\langle 0 \rangle_\sigma}\Big).$$

To see the equality, recall that $f = g^{-s}$.

Notice that each party now holds a *multiplicative share* of $(xy \cdot s,\ xy)$, which corresponds to the party having the correct memory share "in the exponent" of the group. The next step is converting this back to a subtractive share via the DDLog procedure described above.

Step II: Conversion to memory shares. By applying the DDLog procedure to each component of the above multiplicative share, the parties locally recover subtractive shares of $(xy \cdot s,\ xy)$, i.e., a memory share of xy. To see this, it suffices to observe that:

$$\Big(\mathsf{DDLog}(h^{\langle xy \cdot s \rangle_\sigma} g^{\langle 0 \rangle_\sigma}),\ \mathsf{DDLog}(h^{\langle xy \rangle_\sigma} g^{\langle 0 \rangle_\sigma})\Big) = \Big(\langle xy \cdot s \rangle_\sigma,\ \langle xy \rangle_\sigma\Big) = \langle\!\langle xy \rangle\!\rangle_\sigma.$$

At this point, the parties hold memory shares of the desired product, and can continue multiplying other input shares with the newly derived memory share. This enables the computation of RMS programs, as we briefly explain next.

Computing RMS programs. Observe that if the parties are additionally given memory shares of 1 (e.g., as part of the correlated setup), then they can locally convert any input share into a memory share by computing a multiplication by 1. All in all, this is now sufficient to evaluate the three operations required for RMS programs: (1) An input can be converted to a memory value, (2) an input can be multiplied by a memory value, and (3) any two memory values can be added together to provide a memory value of the sum.

With the above template for how to construct HSS for RMS programs, we are now ready to list some of the challenges and pitfalls associated with constructing *multi-key* HSS.

2.3 Challenges associated with multi-keyness

Before we dive in, we emphasize that the problem of eliminating the correlated setup comes down to two things. First, the parties need to obtain a memory sharing of 1 under some joint secret key derived on the fly. Second, they need a way to obtain input shares encrypted under this joint key. If these two problems were magically resolved, then the computation of RMS programs follows.

In particular, the difficulty lies primarily in getting a "re-encryption" of an HSS input share under some joint key, *without* any interaction or correlated setup.[10] We call this the problem of *synchronizing* input shares. To understand this better, consider two input shares generated as in Equation (1) but defined under two *independent* public keys $f_A := g^{-s_A}$ and $f_B := g^{-s_B}$. In particular, consider the input shares generated by each party independently:

$$\text{Party-}A\text{'s input share: } \Big((g^{r_A},\ h^{x \cdot s_A} f_A^{r_A}),\ (g^{r'_A},\ h^x f_A^{r'_A})\Big)$$
$$\text{Party-}B\text{'s input share: } \Big((g^{r_B},\ h^{y \cdot s_B} f_B^{r_B}),\ (g^{r'_B},\ h^y f_B^{r'_B})\Big).$$

[10] We note that a common reference string is still allowed in this model; what we must avoid is any setup that distributes correlated secrets to parties, which also bars solutions in the PKI model.

The problem is that, given the input shares (and public key) of the other party, it is unclear how the parties can evaluate RMS programs over their independent input shares. While one party, say Alice, can give Bob a share of her secret key s_A, which would then allow the two parties to compute an RMS program *using Alice's input shares*, the multi-key problem arises when trying to compute a program using both the inputs of Alice *and* Bob. This is where prior work resorts to an extra round of communication: the parties first agree on a joint public-key in the first round and then share their inputs using this joint key in the second round [BGI17, OSY21, ADOS22]. In the multi-key setting, the question becomes:

How can Alice and Bob non-interactively *obtain a "synchronized" input share under a joint public key?*

Interestingly, this question can be *partially* resolved by leveraging the structure of ElGamal-style encryption. In particular, given Alice's public key f_A, Bob can compute a joint public key $f := f_A \cdot f_B = g^{-(s_A+s_B)}$. Observe that Alice and Bob can actually interpret their own keys as being "trivial" memory shares of 1 under the joint secret key $s = s_A + s_B$, since $(s_A, 1)$ and $(s_B, 0)$ form subtractive shares of $(s, 1)$, satisfying the invariant of Equation (2). Then, for an input share sent by Alice, Bob can compute a "partially synchronized" input share under the joint public key as:

$$\Big((g^{r_A},\ h^{x \cdot s_A} f_A^{r_A} \cdot (g^{r_A})^{-s_B}),\ (g^{r'_A},\ h^{x} f_A^{r'_A} \cdot (g^{r'_A})^{-s_B}) \Big) = \Big((g^{r_A},\ h^{x \cdot s_A} f^{r_A}),\ (g^{r'_A}, h^{x} f^{r'_A}) \Big),$$

which defines a valid ciphertext tuple under the joint secret key.

Moreover, given that Alice generated the input share, she can trivially re-encrypt it on her end under the joint key $f := g^{-(s_A+s_B)}$ and using the same randomness r_A and r'_A (reusing the randomness r_A, r'_A ensures that Alice and Bob obtain the exact same synchronized input share at the end).

This idea *almost* gives a valid input share under the joint public key. The only issue is that the "synchronized" share still has Alice's secret key s_A encrypted in the first component. Unfortunately, while seemingly minor, this is a major obstacle in achieving multi-key HSS. In particular, the above idea fails to give an encryption of $x \cdot (s_A + s_B)$ and thus the resulting ciphertexts do not constitute a valid input share with respect to the joint public key. This prevents the parties from computing RMS programs (indeed, it is not even possible to convert such a share to a memory share, let alone compute a multiplication).

Intuitively, the reason why Alice and Bob are able to synchronize the encryption of x (and not $x \cdot s_A$) is because they can both compute $g^{r'_A \cdot s_B}$: Alice using her knowledge of r'_A and Bob using his knowledge of s_B. Upon closer inspection, this was made possible because *both* r'_A and s_B are random, which means giving out $g^{r'_A}$ and g^{s_B} does not compromise security and makes it possible to compute $(g^{s_B})^{r'_A} = (g^{r'_A})^{s_B}$ *à la* Diffie–Hellman key exchange [DH76].

In contrast, we run into trouble when doing the same with the encryption of $x \cdot s_A$. Getting an encryption of $x \cdot (s_A + s_B)$ seems to require Alice and Bob to compute $h^{x \cdot s_B}$. This seems challenging for two reasons. First, unlike in the previous case, it is insecure to send h^x or h^{s_B} since discrete logarithms are easy over the subgroup generated by h. However, even if we were to use g, Alice cannot send g^x because x is not random and therefore g^x leaks information on x.

To get around these challenges, we take inspiration from constructions of both HSS and multi-key fully-homomorphic encryption (FHE) schemes, and carefully string together several ideas and observations, which we explain next.

Towards full synchronization. First, we find that we can use a trick described by Abram et al. [ADOS22] to avoid "explicitly" encrypting $x \cdot s_A$ in the context of HSS. This technique was used by Abram et al. to obtain circular security, while we observe that it gives us an important stepping stone towards synchronizing input shares. Intuitively, by no longer requiring encryptions of the secret key to be associated with each input share, it becomes easier to define a synchronized ciphertext.

Then, we look for inspiration from the multi-key FHE literature. While the techniques therein do not directly apply to group-based computations, we nonetheless find two key ingredients—"ciphertext expansion" and "encryption of randomness"—to be useful when adapted to a group-based setting. In doing so, we rely on what we will informally refer to as "private homomorphism," which essentially allows two parties to homomorphically evaluate a function using different inputs, while still arriving at identical outputs. Private homomorphism appears to be a unique feature of group-based encryption schemes and does not have an obvious analogue to techniques used to realize multi-key FHE. We provide a detailed explanation of these ideas in the next section.

2.4 Full synchronization

We first start by describing how we can get a step closer to multi-key HSS by avoiding the need for the parties to have encryptions of the secret key as part of the input shares and instead only giving out "implicit" encryptions of the key.

Idea I: Use "flipped" ElGamal. Abram et al. [ADOS22] observe that it is possible to define a "flipped" ElGamal-like encryption by reversing the role of g and the public key in a ciphertext. A surprising feature of flipped encryption is that "input-to-memory conversion" automatically yields a subtractive share of $x \cdot s$ when decrypted with a share of the correct decryption key s. In more detail, the idea is that if Alice generates her input share as:

$$[\![x]\!]_A := \Big((h^x g^{r_A},\ f_A^{r_A}),\ (g^{r'_A},\ h^x f_A^{r'_A}) \Big),$$

then the first (highlighted) component can only be decrypted by computing $(h^x g^{r_A})^{s_A} \cdot (f_A^{r_A}) = h^{x \cdot s_A}$, which corresponds to a decryption of the desired result. (Note that it is still possible to decrypt h^x in the usual way using the second ciphertext present in the input share.)

Now observe that attempting to decrypt the first component using the joint secret key $s = s_A + s_B$ we defined earlier, gives[11]

$$(h^x g^{r_A})^s \cdot (f_A^{r_A}) = h^{x \cdot s} \cdot g^{r_A \cdot s_B}. \tag{3}$$

Observe that this is almost what we want, i.e., we obtain $h^{x \cdot s}$ except it is masked by an extra "junk term" $g^{r_A \cdot s_B}$. Peikert and Shiehian [PS16] describe a similar "junk term" in the context of multi-key FHE decryption. To synchronize Alice's input share under the joint key, parties need to remove this junk term by computing $g^{-r_A \cdot s_B}$. While it is not immediately clear how parties can compute this, it does seem to remove the challenges we faced with our previous attempt at synchronization in Section 2.3. In particular, (1) the product $-r_A \cdot s_B$ is now being computed in the exponent of g instead of h and (2) the product is between r_A and s_B, both of which are random.

At this point, it would seem like we have a potential solution for synchronization by following the same approach we used to synchronize the encryption of x in Section 2.3. Namely, Alice additionally sends g^{r_A} so that both parties can compute $g^{-r_A \cdot s_B}$. However, this approach is completely insecure, since in the flipped encryption variant, g^{r_A} is used to mask h^x. In particular, we note that this issue stems from using flipped ElGamal making our approach for synchronization with the standard ElGamal formulation no longer apply. In other words, it seems like using flipped ElGamal enables circumventing the problem of encrypting the secret key but brings us to another apparent impasse:

How can the parties securely remove the
extra junk term from the decryption process?

Idea II: Encrypt the randomness used for encryption. Coming back to the partial synchronization of the message x described in Section 2.3, we recall that it works because $(g^{s_B})^{r'_A} = (g^{r'_A})^{s_B}$. At its core, this unique homomorphism allows each party to compute a *private function* using the *public encoding* of the other party's input such that both parties finally end up with the same output. Intuitively, it appears that synchronization requires exploiting such a property. Can a similar equation be computed while keeping g^{-r_A} private?

We observe that this is indeed possible by exploiting the homomorphism properties of the encryption scheme: instead of sending g^{r_A} in the clear to Bob, Alice *encrypts* g^{r_A} as $(g^u, g^{r_A} \cdot f_A^u)$, where u is fresh randomness sampled for encrypting g^{r_A}. Now, observe that Alice and Bob can compute an encryption of $g^{-r_A \cdot s_B}$ under Alice's public key f_A as follows.

$$\begin{aligned}
\text{Bob computes:}\ & \Big((g^u)^{-s_B}, (g^{r_A} \cdot f_A^u)^{-s_B} \Big) && = \left(g^{-s_B \cdot u}, g^{-r_A \cdot s_B} \cdot f_A^{-s_B \cdot u} \right). \\
\text{Alice computes:}\ & \Big((f_B)^u, (f_B)^{r_A} \cdot (f_B)^{-s_A \cdot u} \Big) && = \left(g^{-s_B \cdot u}, g^{-r_A \cdot s_B} \cdot f_A^{-s_B \cdot u} \right).
\end{aligned}$$

Note that Bob exploits the additive homomorphism of the encryption scheme to multiply the encrypted message g^{r_A} with his secret key $-s_B$. On the other hand, Alice, essentially replaces the

[11] We note that neither Alice nor Bob can actually carry out this decryption. We are only interested in the structure of the ciphertext here.

generator g with Bob's public key f_B and uses the randomness u to re-encrypt the randomness r_A. This generalizes the idea of having Alice and Bob run private functions on the public encodings of the other party's input to arrive at identical outputs.

While we seem to have made progress on our initial goal of computing $g^{-r_A \cdot s_B}$ in masked form, it is still unclear if this can help synchronize Alice's input shares under some joint key. In particular, this encryption of the junk term can *only* be decrypted under Alice's secret key, and this is a requirement imposed by semantic security.[12] Can we nonetheless define the joint key in a way that still enables decrypting the junk term? Surprisingly, we find that the answer is yes, and we achieve it by exploiting the linearity of the decryption procedure.

Linearity of decryption. Inspired by the multi-key FHE constructions [LTV12, MW16, PS16], we define the joint secret key as being a *concatenation* (rather than a sum) of the two individual keys. The motivation for doing is quite natural in hindsight: observe that summing the keys completely destroys the information required to decrypt a ciphertext generated individually under each key. However, in the concatenated approach, the keys are information-theoretically preserved. This is helpful because it still allows computing both the "wrong" decryption (masked by the junk term, just using Bob's key) *and* a "correct" decryption of the junk term (just using Alice's key). Then, consider concatenation of a "synchronized" ciphertext and flipped ElGamal ciphertext, which we will denote by $\mathbf{c}$, and which is the form

$$\mathbf{c} := \Big((g^{-u \cdot s_B},\ g^{-r_A \cdot s_B} g^{s_A \cdot s_B \cdot u} f_A^{-r_A}),\ (h^x g^{r_A},\ f_A^{r_A}) \Big).$$

We can extend the decryption procedure in the natural way so that the former is decrypted just using Alice's share $\mathbf{s}_A := (s_A, 1, 0, 0)$ of the concatenated key while the latter is decrypted just using Bob's share $\mathbf{s}_B := (0, 0, s_B, 1)$. Then, by viewing decryption as an inner product "in the exponent" with the decryption key $\mathbf{s} = (s_A, 1, s_B, 1)$ which, by abusing notation we will denote as $\langle \mathbf{c}, \mathbf{s} \rangle$, we see that the junk terms cancel out:

$$\begin{aligned} \langle \mathbf{c}, \mathbf{s} \rangle =& (g^{-u \cdot s_B})^{s_A} \cdot (g^{-r_A \cdot s_B} g^{s_A \cdot s_B \cdot u} f_A^{-r_A})^1 \cdot (h^x g^{r_A})^{s_B} \cdot (f_A^{r_A})^1 \\ =&\ g^{-u \cdot s_B \cdot s_A} \cdot g^{-r_A \cdot s_B} g^{s_A \cdot s_B \cdot u} g^{s_A \cdot r_A} \cdot h^{x \cdot s_B} g^{r_A \cdot s_B} \cdot g^{-s_A \cdot r_A} \\ =&\ g^{\cancel{-u \cdot s_B \cdot s_A}} \cdot g^{\cancel{-r_A \cdot s_B}} g^{\cancel{s_A \cdot s_B \cdot u}} g^{\cancel{s_A \cdot r_A}} \cdot h^{x \cdot s_B} g^{\cancel{r_A \cdot s_B}} \cdot g^{\cancel{-s_A \cdot r_A}} = h^{x \cdot s_B}. \end{aligned} \tag{4}$$

Finally, note that when the parties use their shares of the secret key, they obtain a multiplicative share of $h^{x \cdot s_B}$; this multiplicative share can then be converted to subtractive shares of $x \cdot s_B$ using the DDLog procedure as in standard HSS constructions.

Stepping back, it is useful to observe that we've done nothing "illegal" here. We have simply (1) transformed public encryptions of a message x into an expanded ciphertext and (2) defined an joint secret key $\mathbf{s}$ with respect to which it decrypts.

In what follows, we show that this approach works to fully synchronize both Alice's and Bob's input shares under the common secret key $\mathbf{s} = (s_A, 1, s_B, 1)$ that is implicitly defined by each party's "extended" secret key share, and in turn provides a way to evaluate RMS programs on the joint input.

Defining multi-key HSS input shares. Now that we've changed the definition of the secret key, we need to also update the way in which the HSS input shares are defined, so as to ensure we can still compute the decryption as an "inner product in the exponent" using the concatenated secret key. Doing so is trivial and can be achieved by defining an extended ciphertext of the form:

$$\Big(\underbrace{(h^x g^{r_A},\ f_A^{r_A},\ g^0,\ g^0)}_{\text{Ciphertext 1}},\ \underbrace{(g^{r'_A},\ h^x f_A^{r'_A},\ g^0,\ g^0)}_{\text{Ciphertext 2}} \Big),$$

where the extra (highlighted) components enable us to concisely define the decryption as an inner product "in the exponent" with the concatenated secret key $\mathbf{s}$, as in Equation (4).

[12] Indeed, suppose that Alice and Bob could obtain multiplicative shares of $g^{r_A \cdot s_B}$ using shares of the joint secret key $s = -(s_A + s_B)$. Then, this means that s removes the randomness used to encrypt r_A. However, given that s_B is random and independent of s_A, this actually means that it was possible to remove the randomness with a uniformly random secret key, ergo the encryption scheme is not semantically secure.

2.5 Putting everything together

Here we summarize the construction and show that the parties recover subtractive shares of x and $x \cdot s_A$, in addition to $x \cdot s_B$ (as was already shown above). Together, these three values form a complete HSS memory share of x with respect to the joint secret key $\mathbf{s}$. To wit, we define the synchronized input shares of Alice's message x, denoted $\{\!\{x\}\!\}$, as the set of four ciphertext vectors ($\mathbf{c}_1^A$, $\mathbf{c}_2^A$, $\mathbf{c}_1^B$, $\mathbf{c}_2^B$) that respectively decrypt to $(x \cdot s_A,\ x,\ x \cdot s_B,\ x)$ "in the exponent" via an inner product with $\mathbf{s}$:

$$\mathbf{c}_1^A = (h^x g^{r_A},\ g^{-s_A \cdot r_A},\ g^0,\ g^0), \qquad \mathbf{c}_1^B = (g^{-u \cdot s_B},\ g^{-r_A \cdot s_B} g^{s_A \cdot s_B \cdot u} g^{s_A \cdot r_A},\ h^x g^{r_A},\ g^{-s_A \cdot r_A}),$$

$$\mathbf{c}_2^A = (g^{r'_A},\ h^x g^{-s_A \cdot r'_A},\ g^0,\ g^0), \qquad \mathbf{c}_2^B = (g^{r'_A},\ h^x g^{-s_A \cdot r'_A},\ g^0,\ g^0).$$

First, we note that both Alice and Bob can derive ($\mathbf{c}_1^A$, $\mathbf{c}_2^A$, $\mathbf{c}_1^B$, $\mathbf{c}_2^B$) from an input share of Alice's message x (we've already shown this above for $\mathbf{c}_1^B$ and the other cases are easier to derive). Now, given this set of vectors, it becomes easy for Alice and Bob to recover subtractive shares of $(x \cdot s_A, x, x \cdot s_B, x)$, which forms a memory share of x under the concatenated key $\mathbf{s}$. To see this, observe that:

$$\begin{aligned}
\langle \mathbf{c}_1^A, \mathbf{s} \rangle &= h^{x \cdot s_A} g^{r_A \cdot s_A} \cdot g^{-s_A \cdot r_A} \cdot g^0 \cdot g^0 = h^{x \cdot s_A} \\
\langle \mathbf{c}_2^A, \mathbf{s} \rangle &= g^{r'_A \cdot s_A} \cdot h^x g^{-s_A \cdot r'_A} \cdot g^0 \cdot g^0 = h^x \\
\langle \mathbf{c}_1^B, \mathbf{s} \rangle &= g^{-u \cdot s_B \cdot s_A} \cdot g^{-r_A \cdot s_B} g^{s_A \cdot s_B \cdot u} g^{s_A \cdot r_A} \cdot h^{x \cdot s_B} g^{r_A \cdot s_B} \cdot g^{-s_A \cdot r_A} = h^{x \cdot s_B} \\
\langle \mathbf{c}_2^B, \mathbf{s} \rangle &= g^{r'_A \cdot s_A} \cdot h^x g^{-s_A \cdot r'_A} \cdot g^0 \cdot g^0 = h^x,
\end{aligned}$$

where we recall that $\langle \cdot, \cdot \rangle$ denotes computing the inner product "in the exponent."

It then follows that given secret shares of $\mathbf{s}$ (which is now equivalent to a memory share of 1), the parties can obtain multiplicative shares of $(h^{x \cdot s_A},\ h^x,\ h^{x \cdot s_B},\ h^x)$, which they can locally convert into subtractive shares of $(x \cdot s_A,\ x,\ x \cdot s_B,\ x)$ via the DDLog procedure. In turn, this is a valid HSS memory share of x under the secret key $\mathbf{s}$.

In conclusion, this achieves our starting goal of letting the parties obtain an HSS input share synchronized under a joint secret key. At a high level, all the invariants required for evaluating RMS programs are maintained given that: (1) an input share from each party can be converted to a memory share defined with respect to the joint secret key, (2) an input share can still be multiplied by a memory share given that we've preserved the linear decryption property, and (3) memory shares remain additively homomorphic.

Tying up loose ends. While we've described everything from Alice's perspective, synchronizing an input share provided by Bob follows a symmetric sequence of steps. In particular, we highlight that we describe memory shares as four-tuples $(x \cdot s_A, x, x \cdot s_B, x)$, which allows multiplying with a synchronized input share provided by either party (since the corresponding "slot" gets decrypted via the inner product with the secret key). Once both parties have synchronized their respective shares, they have all the necessary ingredients to evaluate RMS programs over their joint inputs.

Finally, as mentioned in the beginning of the overview, the above description glosses over several important details in the interest of clarity. In particular, we need to be careful about what space each message and secret key is defined in. In order to evaluate RMS programs, we need to ensure that the computations (multiplication, addition) do not "wrap around" the order of h. Solving this requires us to (1) define the message space to be integers bounded in absolute value by some bound B and (2) ensure that we can multiply the messages by the secret key without overflow. However, these points are not unique to our constructions and the standard solutions from public-key HSS constructions (e.g., [BGI16, OSY21, RS21]) apply to our constructions too.

Building multi-key HSS in the NIDLS framework. The non-interactive discrete logarithm sharing (NIDLS) framework gives us group $\mathbb{G}$, a generator g in which the discrete logarithm is assumed to be computationally intractable, and a generator h (for a subgroup of $\mathbb{G}$) where the discrete logarithm is efficiently computable. An example of such a group $\mathbb{G}$ is the Paillier [Pai99] group $\mathbb{Z}_{N^2}^*$, where g is a random generator of $\mathbb{Z}_{N^2}^*$, and $h := (N + 1)$ is a generator for a subgroup of order N. However, the most important part of the framework is that it also gives an efficient DDLog algorithm base h. Combined, this gives us all the necessary ingredients to realize multi-key HSS.

Our multi-key HSS construction in the NIDLS framework, presented in Section 4.3, is nearly identical to the construction overviewed in Section 2.4. The main differences are with respect to setting HSS parameters so as to ensure correctness, which we do by following prior work. In particular, for correctness and security, we make the short-exponent assumption (which makes it possible to have short secret keys that fit in the message space) and make the DDH assumption (over the Paillier group or over class groups, depending on the NIDLS instantiation).

Informal Theorem 6 (Multi-key HSS from NIDLS)**.** *Under the DDH and short-exponent assumptions in the Paillier group $\mathbb{Z}_N$ or class groups, there exists a two-party, multi-key homomorphic secret sharing scheme for computing any polynomial-size RMS program, with a negligible correctness error.*

2.6 Extending the ideas to the DDH setting

We now turn our attention to constructing MKHSS from the DDH assumption over any prime-order cyclic group $\mathbb{G}$. At first glance it may appear simple to adapt the construction from Section 2.4 to the DDH setting. It turns out, however, that a new set of ideas is required. The primary roadblock we face in the DDH setting is that, unlike in the NIDLS framework, there is no DDLog procedure for large messages. In DDH-hard groups, DDLog is only suitable for computing a distributed discrete logarithm for *small* messages and, moreover, has a tuneable $1/\mathsf{poly}$ correctness error [BGI16, DKK18]. This prevents us from using the flipped ElGamal approach directly, since the parties would recover multiplicative shares of $g^{x\cdot s}$ with no way to obtain subtractive shares of $x \cdot s$. In Section 2.6.1, we briefly recall how HSS constructions are realized under DDH. Then, in Section 2.6.2, we highlight the challenges faced in adapting the ideas in our NIDLS-based construction to the DDH setting.

2.6.1 Background: HSS from DDH via BHHO Here, we give a brief overview of how public-key HSS can be realized under DDH using the BHHO encryption scheme. We focus on the BHHO-based variant (rather than ElGamal) because it offers several advantages for realizing *multi-key* HSS under DDH, as will become apparent later in Section 2.6.2.

In a nutshell, the BHHO scheme can be seen as a "bit-wise" extension of the ElGamal encryption scheme and is defined with respect to $\ell_{\mathsf{sk}} + 1$ random generators $(g_1, \dots, g_{\ell_{\mathsf{sk}}}, g)$ from the DDH-hard group. The secret key $s := (s_1, \dots, s_{\ell_{\mathsf{sk}}})$ is an ℓ_{sk}-length vector of *bits* and the public key f is defined as: $f := \prod_{i=1}^{\ell_{\mathsf{sk}}} g_i^{-s_i}$. The encryption of a message x under f is then defined as: $(g_1^r, \dots, g_{\ell_{\mathsf{sk}}}^r,\ g^x f^r)$.

Correspondingly, the BHHO-based HSS input share of a message x is defined as an encryption of x along with all the encryptions of the $x \cdot s_i$, for all $i \in [\ell_{\mathsf{sk}}]$:

$$[\![x]\!]_\sigma := \Big(\underbrace{g_1^{r_1}, \dots, g_{\ell_{\mathsf{sk}}}^{r_1},\ g^{x\cdot s_1} f^{r_1}, \dots, g^{x\cdot s_{\ell_{\mathsf{sk}}}} f^{r_{\ell_{\mathsf{sk}}}}, g_1^{r_{\ell_{\mathsf{sk}}+1}}, \dots, g_{\ell_{\mathsf{sk}}}^{r_{\ell_{\mathsf{sk}}+1}}, g^x f^{r_{\ell_{\mathsf{sk}}+1}}}_{\ell_{\mathsf{sk}}+1 \text{ Ciphertexts}}\Big).$$

To compute a multiplication with a memory share, the idea is to first compute the component-wise multiplication between the ciphertexts encrypting $(s_1, \dots, s_{\ell_{\mathsf{sk}}}, 1) \cdot x$ and the memory share of y, which consists of subtractive shares of $(s, 1) \cdot y$. Let $s_{\ell_{\mathsf{sk}}+1} := 1$ for notational convenience. Observe that, for all $i \in [\ell_{\mathsf{sk}} + 1]$, we can decrypt $x \cdot s_i$ using the secret key s by computing:

$$g^{x\cdot s_i} f^{r_i} \cdot \prod_{j=1}^{\ell_{\mathsf{sk}}} (g_j^{r_i})^{s_j} = g^{x\cdot s_i}.$$

It follows that if party-σ is given a memory share of the form $(\langle y \cdot s\rangle_\sigma, \langle y\rangle_\sigma)$, they can recover a multiplicative share of the form $g^{\langle xy\cdot s_i\rangle_\sigma}$, for all $i \in [\ell_{\mathsf{sk}} + 1]$, by exploiting the exponent-linear decryption property. Then, because $xy \cdot s_i$ is small, the two parties can recover subtractive shares of $xy \cdot s_i$ using the DDLog procedure. Finally, the parties hold subtractive shares of the vector $(s_1, \dots, s_{\ell_{\mathsf{sk}}}, 1) \cdot xy$, which corresponds to a memory share under the BHHO secret key.

2.6.2 Recovering the implicit encryptions As discussed in Section 2.3, the fundamental challenge in the multi-key setting is to non-interactively synchronize HSS input shares under a joint key. In Section 2.4, we showed that using a flipped encryption helps make the problem tractable: it provides an implicit encryption of $x \cdot s_B$ (when decrypted with Bob's secret key). Fortunately, we find that we can emulate the same process under BHHO (and indeed, such a flipped encryption was even used to prove the circular security of the BHHO scheme [BHHO08]).

Flipped encryption under BHHO. The BHHO ciphertexts allows us to define the analogous flipped encryption we exploited in the NIDLS framework to realize synchronization. Observe that if we put the message x "in the wrong place" and encrypt it as:

$$\mathsf{ct}_i := (g_1^r, \ldots, g_{i-1}^r, g^x g_i^r, g_{i+1}^r, \ldots, g_{\ell_{\mathsf{sk}}}^r,\ f^r),$$

then we have that the decryption procedure—which simply computes an inner product "in the exponent" with the secret key $s = (s_1, \ldots, s_{\ell_{\mathsf{sk}}})$—produces:

$$(g^x)^{s_i} \cdot f^r \cdot \prod_{j=1}^{\ell_{\mathsf{sk}}} (g_j^r)^{s_j} = g^{x \cdot s_i},$$

which gives us an "implicit" encryption of the i-th bit of s.

With this, we've recovered the first stepping stone required for replicating our NIDLS-based multi-key HSS construction outlined in Section 2.4. We now turn to recovering the other properties we exploited in the NIDLS-based construction.

Applying the multi-key template. As in the case of the NIDLS construction, using a flipped encryption of Alice's input x simplifies computing the product $x \cdot s_B$ in the exponent, but comes at the cost of having to account for, and later negate, the resulting "junk" term. In particular, upon decrypting $\mathsf{ct}_i := (g_1^{r_A}, \ldots, g_{i-1}^{r_A}, g^x g_i^{r_A}, g_{i+1}^{r_A}, \ldots, g_{\ell_{\mathsf{sk}}}^{r_A},\ f_A^{r_A})$ using Bob's secret key we get:

$$g^{x \cdot s_B^{(i)}} \cdot f_A^{r_A} \cdot \prod_{i=1}^{\ell_{\mathsf{sk}}} g_i^{r_A \cdot s_B^{(i)}} = g^{x \cdot s_B^{(i)}} \cdot \overbrace{f_B^{-r_A} \cdot f_A^{r_A}}^{\text{junk term}}.$$

Recall that our MKHSS construction from Section 2.4 circumvents this issue by computing a synchronized encryption of the junk term under Alice's secret key. It then exploits the linearity of decryption to concatenate the ciphertexts and secret keys such that the junk term gets decrypted by Alice's secret key and negates the junk term created by decrypting with Bob's secret key. Clearly, the BHHO decryption procedure is also linear, and thus appears amenable to this idea as well. However, it is not clear how Alice and Bob can compute a synchronized encryption of the junk term under Alice's secret key, given that the secret keys are now bits, which complicates exploiting the linear homomorphism. This is where we need a new set of ideas.

2.7 Recovering full synchronization under BHHO

We revisit the primary goal of synchronization. As alluded to in Section 2.4, synchronization requires each party to compute a *private function*, using the *public encoding* of the other party's input share, such that *both* parties arrive at the same synchronized output. The reason why BHHO makes this difficult is that the public key is defined as an *inner product* ("in the exponent") between the secret key and the group elements $(g_1, \ldots, g_{\ell_{\mathsf{sk}}})$. This provides some intuition as to why our previous attempt fails: we need to find a way of *publicly* encoding Alice's randomness r_A such that Bob can *privately* compute the inner product function with his secret key. Another way of seeing this is that we need to extend the ideas from the NIDLS-based construction from privately computing a product to privately computing an inner product.

Using this intuition, we observe that Bob already can compute $f_B^{-r_A}$,[13] as long as Alice's encoding additionally includes $(g_1^{r_A}, \ldots, g_{\ell_{\mathsf{sk}}}^{r_A})$, since he can evaluate:

$$\prod_{i=1}^{\ell_{\mathsf{sk}}} g_i^{r_A s_B^{(i)}} = f_B^{-r_A}.$$

[13] We focus on $f_B^{-r_A}$, since the other factor of the junk term (i.e., $f_A^{r_A}$) is public and available to both the parties.

This idea provides a way forward. As a first attempt at using this observation, we let Alice encrypt each $g_i^{r_A}$ using her public key. That is, she defines her input share to be the list of ℓ_{sk} BHHO ciphertexts, each encrypting the *same* r_A but crucially under *different* randomness $u_1, \ldots, u_{\ell_{\mathsf{sk}}}$:

$$\begin{bmatrix} g_1^{u_1} & g_2^{u_1} & \cdots & g_{\ell_{\mathsf{sk}}}^{u_1} & g_1^{r_A} \cdot f_A^{u_1} \\ g_1^{u_2} & g_2^{u_2} & \cdots & g_{\ell_{\mathsf{sk}}}^{u_2} & g_2^{r_A} \cdot f_A^{u_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_1^{u_{\ell_{\mathsf{sk}}}} & g_2^{u_{\ell_{\mathsf{sk}}}} & \cdots & g_{\ell_{\mathsf{sk}}}^{u_{\ell_{\mathsf{sk}}}} & g_{\ell_{\mathsf{sk}}}^{r_A} \cdot f_A^{u_{\ell_{\mathsf{sk}}}} \end{bmatrix}$$

Given this matrix of group elements, Bob can compute an inner product between each column and his secret key to obtain the ciphertext vector:

$$(g_1^{s^*}, \ldots, g_{\ell_{\mathsf{sk}}}^{s^*}, f_B^{-r_A} \cdot f_A^{s^*}),$$

where $s^* = \sum_i s_B^{(i)} \cdot u_i$. Observe that this corresponds exactly to an encryption—under Alice's public key and with randomness s^*—of the junk term $f_B^{-r_A}$ we need. Thus, it may appear that we found a way for Alice to securely encode her randomness r_A such that Bob can apply the private inner product function, defined by his secret key, and compute an encryption of the junk term that is decryptable under Alice's secret key.

Unfortunately, we are still left with one small problem. While above we've shown how Bob can hypothetically synchronize, Alice is unable to compute the identical ciphertext computed by Bob on her end since it requires her to know s^*, which in turn is a linear function of Bob's secret key and hence cannot be given out.

Randomness reuse to the rescue. To avoid having the term s^* altogether, we can try to have Alice *reuse* the same randomness u for all encryptions of $g_i^{r_A}$. That is, if she instead encodes her randomness r_A using the following matrix:

$$\begin{bmatrix} g_1^{u} & g_1^{u} & \cdots & g_1^{u} & g_1^{r_A} \cdot f_A^{u} \\ g_2^{u} & g_2^{u} & \cdots & g_2^{u} & g_2^{r_A} \cdot f_A^{u} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_{\ell_{\mathsf{sk}}}^{u} & g_{\ell_{\mathsf{sk}}}^{u} & \cdots & g_{\ell_{\mathsf{sk}}}^{u} & g_{\ell_{\mathsf{sk}}}^{r_A} \cdot f_A^{u} \end{bmatrix},$$

then the column-wise inner product with Bob's secret key would result in all the u's "factoring-out" leading to Bob obtaining: $(f_B^u, \ldots, f_B^u, f_B^{-r_A} \cdot f_A^{u \cdot \sum_i s_B^{(i)}})$.

While this would now allow Alice to synchronize too, re-using randomness in this way compromises the security of r_A. Moreover, even with this change, Alice can only compute the first $\ell_{\mathsf{sk}} - 1$ components. It is unclear how she can synchronize the last component: Alice needs to compute $f_A^{u \cdot \sum_i s_B^{(i)}}$ using f_B.

The final observation we make allows us to address both of these issues simultaneously, while still getting the benefits of the randomness-reuse attempt above. By more closely examining the reason why Alice cannot synchronize her last component, it becomes clear that her public key f_A is *itself* an inner product ("in the exponent") computed between her secret key $(s_A^{(1)}, \ldots, s_A^{(\ell_{\mathsf{sk}})})$ and $(g_1, \ldots, g_{\ell_{\mathsf{sk}}})$. Now, if instead we had the last component of each ciphertext be $(g_1^{r_A} \cdot g_1^{\Gamma}, \ldots, g_{\ell_{\mathsf{sk}}}^{r_A} \cdot g_{\ell_{\mathsf{sk}}}^{\Gamma})$, where Γ is some random mask that is known to Alice, then Bob would obtain $f_B^{r_A} \cdot f_B^{\Gamma}$ upon computing the inner product. Moreover, using Γ allows Alice to compute f_B and synchronize on her end. However, Γ needs to be chosen carefully, since parties should still be able to synchronize to an encryption of the junk term under Alice's secret key s_A. With this in mind, we observe that Alice can sample a vector $(\gamma_1, \ldots, \gamma_{\ell_{\mathsf{sk}}}) \leftarrow\$ \mathbb{Z}_p^{\ell_{\mathsf{sk}}}$ uniformly at random and set $\Gamma := \sum_i s_A^{(i)} \cdot \gamma_i$. Then, she can encrypt the randomness r_A as:

$$\begin{bmatrix} g_1^{u\cdot\gamma_1} & g_1^{u\cdot\gamma_2} & \cdots & g_1^{u\cdot\gamma_{\ell_{\mathsf{sk}}}} & g_1^{r_A} \cdot g_1^{u\cdot\Gamma} \\ g_2^{u\cdot\gamma_1} & g_2^{u\cdot\gamma_2} & \cdots & g_2^{u\cdot\gamma_{\ell_{\mathsf{sk}}}} & g_2^{r_A} \cdot g_2^{u\cdot\Gamma} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_{\ell_{\mathsf{sk}}}^{u\cdot\gamma_1} & g_{\ell_{\mathsf{sk}}}^{u\cdot\gamma_2} & \cdots & g_{\ell_{\mathsf{sk}}}^{u\cdot\gamma_{\ell_{\mathsf{sk}}}} & g_{\ell_{\mathsf{sk}}}^{r_A} \cdot g_{\ell_{\mathsf{sk}}}^{u\cdot\Gamma} \end{bmatrix},$$

which can be proven to hide r_A under the matrix-DDH assumption (which is implied by DDH). It is then not hard to see that by computing the synchronization as before, Bob gets:

$$(f_B^{-u\cdot\gamma_1}, \dots, f_B^{-u\cdot\gamma_{\ell_{\mathsf{sk}}}}, f_B^{-r_A} \cdot f_B^{-u\cdot\Gamma}),$$

and now Alice is also able to synchronize to the same ciphertext too, since she receives f_B and knows r_A, u, *and* Γ. This forms the final synchronized ciphertext in our BHHO-based multi-key HSS construction.

Putting everything together. To recap, Alice computes ℓ_{sk} BHHO public keys $\{(g_i^{\gamma_1}, \dots, g_i^{\gamma_{\ell_{\mathsf{sk}}}}, g_i^{\Gamma})\}_i$, all using the same secret key $(s_A^{(1)}, \dots, s_A^{(\ell_{\mathsf{sk}})})$, such that the i-th component of each public key has the same random value γ_i. Then, under the concatenated decryption, the random value "factors-out" when Bob computes an inner product, leaving $f_B^{\gamma_i}$, which Alice can compute using γ_i and Bob's public key. Repeating this for all keys then allows Alice and Bob to compute a valid memory share with respect to their joint concatenated BHHO secret keys.

Thus, Alice and Bob are able to use the public encoding of the other party's input to compute the same ciphertext, encrypting the junk term under the other party's secret key. It is then easy to see that this is sufficient for evaluating RMS programs, where we use the concatenation of the secret keys as in the NIDLS construction. We give the full construction in Section 4.4, where we prove:

Informal Theorem 7 (Multi-key HSS from DDH)**.** *Assume that the DDH assumption holds in any cyclic group $\mathbb{G}$. Then, there exists a two-party, multi-key homomorphic secret sharing scheme for computing any polynomial-size RMS program, with $1/\mathsf{poly}$ correctness error.*

3 Preliminaries

In this section, we cover the notation that we will use throughout the paper.

General notation. We let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{Z}$ denote the set of integers, $\mathbb{G}$ denote a finite group, and $\mathcal{R}$ denote a finite ring. A reduction modulo t, for any positive integer t, yields a representative in the range $\mathbb{Z}_t = \{-\lfloor t/2 \rfloor, \dots, \lfloor (t-1)/2 \rfloor\}$. We denote by $\mathsf{poly}(\cdot)$ the set of all polynomials and by $\mathsf{negl}(\cdot)$ any negligible function. We occasionally abuse notation and let poly denote a fixed polynomial.

Vectors and matrices. We denote a vector $\mathbf{v}$ using bold lowercase letters and a matrix $\mathbf{A}$ using bold uppercase letters. The i-th coordinate of a vector $\mathbf{v}$ is denoted by $\mathbf{v}[i]$. We will occasionally write $(v_i)_{i=1}^n$ to denote the vector $(v_1, \dots, v_n)$.

Vector group operations. For all $\mathbf{g} \in \mathbb{G}^\ell$ and $\mathbf{x} \in \mathbb{Z}^\ell$, we use $\langle \mathbf{g}, \mathbf{x} \rangle$ to denote $\langle \mathbf{g}, \mathbf{x} \rangle = \prod_{i=1}^\ell g_i^{x_i}$, where $\mathbf{g} = (g_1, \dots, g_\ell)$ and $\mathbf{x} = (x_1, \dots, x_\ell)$.

Sampling and assignment. We let $x \leftarrow\!\!\$\, S$ denote a uniformly random sample drawn from a set S. We let $x \leftarrow \mathcal{A}$ denote assignment from a randomized algorithm $\mathcal{A}$ and $x := y$ denote initialization of x to the value of y (which may be the output of a deterministic algorithm).

Efficiency. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time.

Probability and indistinguishability. We let $\Pr[E : A]$ denote the probability of an event E in an experiment defined by executing A. For two probability ensembles $\{A_i\}_i$ and $\{B_i\}_i$, we use $\{A_i\}_i \equiv \{B_i\}_i$ to denote that the ensembles are identical, $\{A_i\}_i \approx_s \{B_i\}_i$ to denote that the ensembles are statistically close and $\{A_i\}_i \approx_c \{B_i\}_i$ to denote that the ensembles are computationally indistinguishable.

Leftover Hash Lemma. We say a distribution $\mathcal{D}$ over a set $\mathcal{X}$ is ϵ-uniform if $\sum_{x\in\mathcal{X}} \left| \mathcal{D}(x) - \frac{1}{|\mathcal{X}|} \right| \leq \epsilon$. We will make use of the following immediate corollary of the leftover hash lemma that explicitly appears in [BHHO08].

Lemma 1 (Simplified Leftover Hash Lemma [BHHO08, Lemma 2])**.** *Let $\mathcal{H}$ be a family of 2-universal hash functions from a set $\mathcal{X}$ to a set $\mathcal{Y}$. Then, the distribution $(H, H(x))$ where $H \leftarrow\!\!\$\, \mathcal{H}$ and $x \leftarrow\!\!\$\, \mathcal{X}$ is $\sqrt{\frac{|\mathcal{Y}|}{4\cdot|\mathcal{X}|}}$-uniform on $\mathcal{H} \times \mathcal{Y}$.*

Subtractive Sharing. Let $\mathcal{R}$ be a ring. We use $\langle x\rangle^{\mathcal{R}} \in \mathcal{R}^2$ where $\langle x\rangle^{\mathcal{R}} = (\langle x\rangle^{\mathcal{R}}_A, \langle x\rangle^{\mathcal{R}}_B)$ to denote a subtractive sharing of $x \in \mathcal{R}$ such that $\langle x\rangle^{\mathcal{R}}_A - \langle x\rangle^{\mathcal{R}}_B = x$. For ease of notation, we use $\langle x\rangle = (\langle x\rangle_A, \langle x\rangle_B)$ to denote the subtractive sharing over the integers when $\mathcal{R} = \mathbb{Z}$.

Non-Interactive Key Exchange. Here, we provide a basic definition of non-interactive key exchange, which will suffice for our applications and constructions.

Definition 1 (Non-Interactive Key Exchange [DH76, CKS08, FHKP13])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. A non-interactive key exchange (NIKE) scheme consists of algorithms* $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter λ and outputs a common reference string* crs.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$. *The randomized key generation algorithm takes as input the CRS* crs. *It outputs a public key* pk *and secret key* sk.
- $\mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_i, \mathsf{sk}_j) \to K$. *The deterministic key derivation algorithm takes as input the CRS* crs, *a public key* pk_i, *and a secret key* sk_j. *It outputs a key $K \in \{0,1\}^\lambda$.*

The above algorithms must satisfy the following properties:

Correctness. For all security parameters $\lambda \in \mathbb{N}$, it holds that:

$$\Pr\left[K_A = K_B : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ (\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ K_A \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_B, \mathsf{sk}_A) \\ K_B \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \end{array} \right] = 1.$$

Security. For all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[b = b' : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ (\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ K_0 \leftarrow \mathsf{KeyDer}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \\ K_1 \leftarrow\$ \{0,1\}^\lambda \\ b \leftarrow\$ \{0,1\} \\ b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_A, \mathsf{pk}_B, K_b) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

In particular, this security definition for NIKE is known as "CKS-light" security [FHKP13], which is known to be polynomially equivalent to stronger notions of NIKE.

3.1 Distributed evaluation of RMS programs

In this section, we present a unifying template for reasoning about distributed evaluation of HSS input shares, which not only captures HSS evaluation from prior works but will also be useful in proving the correctness of our constructions. Note that our focus here is only on correctness of HSS evaluation, assuming both parties already hold shares of all program inputs. How these inputs are securely shared between the parties will be discussed in subsequent sections.

Restricted Multiplication Straight-line (RMS) programs. We will focus on distributed evaluation of Restricted Multiplication Straight-line (RMS) programs [Cle90, BGI16]. An RMS program is an arithmetic circuit over integers with the restriction that every multiplication is between an input value and an intermediate value of the computation, called a memory value. Most existing HSS schemes support evaluating RMS programs. Boyle et al. [BGI16] show that the class of polynomial-size RMS programs includes the class of polynomial-size branching programs, which is in turn known to contain the class of NC^1 circuits.

Definition 2 (Restricted Multiplication Straight-line Program [Cle90, BGI16]). *A restricted multiplication straight-line (RMS) program P consists of a magnitude bound $B \in \mathbb{N}$ and an arbitrary sequence of the following four instructions.*

- $\mathsf{Convert}(\mathsf{I}_x) \to \mathsf{M}_x$: *Load the value of the input wire I_x to the memory wire M_x.*
- $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y) \to \mathsf{M}_z$: *Add the values of the memory wires M_x and M_y and assign the result to the memory wire M_z.*
- $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y) \to \mathsf{M}_z$: *Multiply the value of the input wire I_x by the value of the memory wire M_y and assign the result to the memory wire M_z.*
- $\mathsf{Output}(\mathsf{M}_z) \to z$: *Output the value of the memory wire M_z.*

If at any step of the execution, the size of a memory value exceeds the bound B, the output of the program on the corresponding input is defined to be $\perp$. The size of an RMS program, denoted by $|P|$, is defined as the number of instructions.

Primitives required for distributed evaluation. The distributed, non-interactive evaluation of RMS programs in group-based HSS schemes rely on two primitives. The first is HSS shares of the inputs, which satisfy a property we abstract as "exponent-linear decoding." This property intuitively captures the decryption process in ElGamal-style public-key encryption schemes instantiated over various groups (e.g., DDH-hard cyclic groups, the Paillier group, class groups, etc.). The second is the distributed discrete logarithm algorithm introduced in [BGI16], which serves as the foundation of all existing group-based HSS constructions. In Lemma 3, we show that these components suffice for distributed evaluation of any RMS program. This framework captures HSS constructions of Boyle et al. [BGI16] from DDH (the BHHO-based scheme), as well as the HSS constructions by Abram et al. [ADOS22] based on either the DCR assumption or DDH-like assumptions in Paillier and class groups. Looking ahead, although inputs in our multi-key HSS constructions are encoded differently from prior works, they still satisfy the exponent-linear decoding property, which in turn allows distributed evaluation of RMS programs.

Definition 3 (Exponent-Linear Decoding). *Let $\mathbb{G}$ be an Abelian group, let $\mathbb{H} \subseteq \mathbb{G}$ be a finite cyclic subgroup of order t with generator h and let $\ell \in \mathbb{N}$. We let $\{\!\{x\}\!\} := (\mathbf{c}_1, \dots, \mathbf{c}_\ell) \in \mathbb{G}^{\ell \times \ell}$ be an encoding of an integer x with base-h exponent-linear decoding under the decoding key $\mathbf{k} = (k_1, \dots, k_\ell) \in \mathbb{Z}^\ell$ if for all $i \in [\ell]$, we have $\langle \mathbf{c}_i, \mathbf{k} \rangle = h^{x \cdot k_i}$.*

Definition 4 (Distributed Discrete Logarithm). *Let $\mathbb{G}$ be an Abelian group, let $\mathbb{H} \subseteq \mathbb{G}$ be a finite cyclic subgroup of order t with generator h, let ε be a real number and B_{dl} be a positive integer, where $0 \leq \varepsilon < 1$ and $B_{\mathsf{dl}} < t$. An efficient algorithm* DDLog *is an ε-correct, B_{dl}-bounded, base-h algorithm for distributed discrete logarithm, if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, all integers x where $|x| \leq B$ and all $f \in \mathbb{G}$ we have*

$$\Pr_{r \leftarrow\$ \{0,1\}^\lambda}[\mathsf{DDLog}(f \cdot h^x; r) - \mathsf{DDLog}(f; r) \not\equiv x \bmod t] \leq \varepsilon + \mathsf{negl}(\lambda).$$

Instantiations of DDLog. We will consider two classes of DDLog algorithms: (1) the DDLog procedure of Boyle et al. [BGI16] (and the improved variant of Dinur et al. [DKK18]) that works over any finite cyclic group and for polynomial-bounded exponents but has an inverse-polynomial correctness error, and (2) DDLog procedures in the Non-Interactive Discrete Log Sharing (NIDLS) framework [ADOS22] that have negligible correctness error and can support super-polynomially bounded exponents. We briefly recall the relevant claim from [BGI16] as well as the definition of the NIDLS framework.

Lemma 2 ([BGI16, Proposition 3.2 and Claim 3.7]). *Let $\mathbb{G}$ be a finite cyclic group of order p. For every polynomial* $\mathsf{poly}(\cdot)$ *and for all $\lambda \in \mathbb{N}$, $\varepsilon > 0$, $B_{\mathsf{dl}} \in \mathbb{N}$, and $g \in \mathbb{G}$, where $1/\varepsilon, B_{\mathsf{dl}} \leq \mathsf{poly}(\lambda) < p$, there exists an ε-correct, B_{dl}-bounded, base-g algorithm for distributed discrete logarithm.*

The NIDLS framework defines a finite Abelian group $\mathbb{G} = \mathbb{H} \times \mathbb{K}$, where the discrete log problem is easy in the cyclic subgroup $\mathbb{H}$ of known order t, and assumed to be computationally intractable in the subgroup $\mathbb{K}$ of unknown order. The framework is equipped with an upper-bound B_{nidls} on the order of $\mathbb{K}$ and an efficiently sampleable distribution $\mathcal{D}_{\mathsf{nidls}}$ over $\mathbb{G}$.

Definition 5 (NIDLS Framework [ADOS22]). *The NIDLS framework consists of three efficient algorithms* $(\mathsf{GGen}, \mathcal{D}_{\mathsf{nidls}}, \mathsf{DDLog})$ *with the following functionality:*

- $\mathsf{GGen}(1^\lambda) \to \mathsf{crs} := (\mathbb{G}, \mathbb{H}, \mathbb{K}, h, t, B_{\mathsf{nidls}}, \mathsf{aux})$. *The randomized group generation algorithm takes as input the security parameter and outputs a common reference string* crs *which consists of:*
  - *finite Abelian group* $\mathbb{G}$,
  - *subgroups* $\mathbb{H}$ *and* $\mathbb{K}$ *such that* $\mathbb{G} = \mathbb{H} \times \mathbb{K}$,
  - *generator* h *and order* t *of* $\mathbb{H}$,
  - *positive integer* B_{nidls},
  - *and auxiliary information* aux.
- $\mathcal{D}(1^\lambda, \mathsf{crs}) \to (f, \rho)$. *The randomized sampling algorithm takes as input the security parameter and common reference string, and outputs a group element* $f \in \mathbb{G}$ *along with some auxiliary information* ρ.
- $\mathsf{DDLog}(\mathsf{crs}, f) =: s$. *The deterministic distributed discrete log algorithm takes as input a common reference string and a group element, and outputs an element* $s \in \mathbb{Z}_t$.

The above functionality needs to satisfy the following properties:

Correctness. For all $\lambda \in \mathbb{N}$ *and efficient adversaries* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[\langle s \rangle_A - \langle s \rangle_B = x \pmod{t} \quad : \quad \begin{aligned} \mathsf{crs} := (\mathbb{G}, \mathbb{H}, \mathbb{K}, h, t, B_{\mathsf{nidls}}, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (f_B, x) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ f_A := h^x \cdot f_B \\ \langle s \rangle_A := \mathsf{DDLog}(\mathsf{crs}, f_A) \\ \langle s \rangle_B := \mathsf{DDLog}(\mathsf{crs}, f_B) \end{aligned} \right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. For all $\lambda \in \mathbb{N}$, *it holds that*

$$\left\{ (\mathsf{crs}, f, \rho, f^r) \;\middle|\; \begin{aligned} \mathsf{crs} := (\mathbb{G}, \mathbb{H}, \mathbb{K}, h, t, B_{\mathsf{nidls}}, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (f, \rho) \leftarrow \mathcal{D}_{\mathsf{nidls}}(1^\lambda, \mathsf{crs}) \\ r \leftarrow\$ [B_{\mathsf{nidls}}] \end{aligned} \right\}$$
$$\approx_s \left\{ (\mathsf{crs}, f, \rho, f') \;\middle|\; \begin{aligned} \mathsf{crs} := (\mathbb{G}, \mathbb{H}, \mathbb{K}, h, t, B_{\mathsf{nidls}}, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (f, \rho) \leftarrow \mathcal{D}_{\mathsf{nidls}}(1^\lambda, \mathsf{crs}) \\ f' \leftarrow\$ \langle f \rangle \end{aligned} \right\}.$$

i.e., the group elements f^r *and* f' *are* statistically *indistinguishable. Note that here,* $\langle f \rangle$ *denotes the group generated by the element* f.

In this work, we will consider instantiations of the NIDLS framework that have a subgroup $\mathbb{H}$ of large order $t > 2^\lambda$, since this is required for distributed evaluation of RMS programs. Known instantiations of the NIDLS framework with a large subgroup $\mathbb{H}$ include the ciphertext space of Paillier and Damgård–Jurik encryption schemes, the ciphertext space of a variant of the Joye–Libert cryptosystem described in [ADOS22], and class groups. We refer to Abram et al. [ADOS22] for a detailed discussion on these instantiations.

Template for distributed evaluation. We conclude this section by describing an algorithm in Figure 1 for distributed evaluation of RMS programs, using PRFs, a DDLog algorithm and encodings of inputs that are exponent-linear decodeable. The proof of correctness closely follows that of group-based HSS constructions in prior works; however, we revisit the details here for completeness.

Lemma 3. *Let* $\mathbb{G}$ *be an Abelian group,* $\mathbb{H} \subseteq \mathbb{G}$ *be a finite cyclic subgroup of order* t *with generator* h, DDLog *be an* ε*-correct,* B_{dl}*-bounded, base-*h *algorithm for distributed discrete logarithm, and* F_1 *and* F_2 *be secure PRFs. Then, for all polynomials* $\mathsf{poly}(\cdot)$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, *all* $\mathbf{k} = (k_1, \ldots, k_{\ell-1}, 1) \in \mathbb{Z}^\ell$, *all* $\mathbf{k}_A \in \mathbb{Z}^\ell$, *all RMS programs* P *with bound* B, *all*

Distributed Evaluation of RMS Program

Public Parameters. Abelian group $\mathbb{G}$ and finite cyclic subgroup $\mathbb{H} \subseteq \mathbb{G}$ of order t with generator h. Base-h distributed discrete logarithm algorithm DDLog. PRF F_1 with output space $\{0,1\}^\lambda$ and a PRF F_2 with output space $\mathbb{Z}_t$.

```
$\mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, (\{\!\{x_1\}\!\}, \ldots, \{\!\{x_m\}\!\}), P)$:
    Parse $\mathsf{ek}_\sigma = (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \langle\!\langle 1 \rangle\!\rangle_\sigma)$.
    For each $\mathsf{id} \in [|P|]$, evaluate the $\mathsf{id}$-th instruction as follows:
      • $\mathsf{Convert}$: $\mathsf{M}_x \leftarrow \mathsf{I}_x$:
        1: Execute the $\mathsf{Mult}(\{\!\{x\}\!\}, \langle\!\langle 1 \rangle\!\rangle_\sigma)$ instruction to compute $\langle\!\langle x \rangle\!\rangle_\sigma$.
      • $\mathsf{Mult}$: $\mathsf{M}_{xy} \leftarrow \mathsf{I}_x \cdot \mathsf{M}_y$:
        1: Parse $\{\!\{x\}\!\} = (\mathbf{c}_1, \ldots, \mathbf{c}_\ell)$.
        2: For $i \in [\ell]$:
          2.1: $f_\sigma^{(i)} \coloneqq \langle \mathbf{c}_i, \langle\!\langle y \rangle\!\rangle_\sigma \rangle$.
          2.2: $\langle z_i \rangle_\sigma \coloneqq \mathsf{DDLog}(f_\sigma^{(i)}; F_1(k_1^{\mathsf{prf}}, \mathsf{id} \| i)) + F_2(k_2^{\mathsf{prf}}, \mathsf{id} \| i) \bmod t$.
        3: $\langle\!\langle xy \rangle\!\rangle_\sigma \coloneqq (\langle z_1 \rangle_\sigma, \ldots, \langle z_i \rangle_\sigma)$.
      • $\mathsf{Add}$: $\mathsf{M}_{x+y} \leftarrow \mathsf{M}_x + \mathsf{M}_y$:
        1: $\langle\!\langle x+y \rangle\!\rangle_\sigma \coloneqq \langle\!\langle x \rangle\!\rangle_\sigma + \langle\!\langle y \rangle\!\rangle_\sigma$.
      • $\mathsf{Output}$: $z \leftarrow \mathsf{M}_z$:
        1: Parse $\langle\!\langle z \rangle\!\rangle_\sigma = (\langle z_1 \rangle_\sigma, \ldots, \langle z_i \rangle_\sigma)$.
        2: Return $\langle z_\ell \rangle_\sigma$.
```

Fig. 1: Distributed evaluation of RMS program.

$x_1, \ldots, x_m \in \mathbb{Z}$ *and* $\{\!\{x_1\}\!\}, \ldots, \{\!\{x_m\}\!\} \in \mathbb{G}^{\ell \times \ell}$*, the algorithm* DEval *described in Figure 1 satisfies*

$$\Pr\left[\langle z \rangle_A - \langle z \rangle_B \neq P(x_1, \ldots, x_m) \quad : \quad \begin{array}{r} k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}} \leftarrow\!\$\ \{0,1\}^\lambda \\ \mathbf{k}_B \coloneqq \mathbf{k}_A - \mathbf{k} \\ \mathsf{ek}_\sigma \coloneqq (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \mathbf{k}_\sigma),\ \forall \sigma \in \{A, B\} \\ \langle z \rangle_\sigma \coloneqq \mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, (\{\!\{x_1\}\!\}, \ldots, \{\!\{x_m\}\!\}), P),\ \forall \sigma \in \{A, B\} \end{array} \right] \leq \varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda),$$

where each $|k_i| \leq B_{\mathsf{sk}}$ *for some* $B_{\mathsf{sk}} \in \mathbb{N}$*, each* $\{\!\{x_i\}\!\}$ *is an encoding of* x_i *with base-*h *exponent-linear decoding under* $\mathbf{k}$*,* $P(x_1, \ldots, x_m) \neq \perp$*,* $\ell \leq \mathsf{poly}(\lambda)$*,* $|P| \leq \mathsf{poly}(\lambda)$*,* $B \cdot B_{\mathsf{sk}} \leq B_{\mathsf{dl}}$ *and* $B \cdot B_{\mathsf{sk}} \cdot 2^\lambda < t$*.*

Proof. Observe that for every memory value M_x in the RMS program, party σ computes a share $\langle\!\langle x \rangle\!\rangle_\sigma$. We must show that the output produced by each party is a subtractive sharing of $P(x_1, \ldots, x_m)$. At a high level, we will show this by proving that DEval maintains the invariant that $\langle\!\langle x \rangle\!\rangle = (\langle\!\langle x \rangle\!\rangle_A, \langle\!\langle x \rangle\!\rangle_B)$ forms a subtractive sharing of $x \cdot \mathbf{k}$, for every memory value M_x. Then, since the last component of $\mathbf{k}$ is 1, the parties obtain a subtractive sharing of the program output upon evaluating DEval.

Note that for $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y)$ instructions, the above invariant holds trivially due to the additive homomorphism of subtractive sharing.

For $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$ instructions, the exponent-linear decoding property allows party-B to compute a $f_B^{(i)} \in \mathbb{G}$ and party-A to compute $f_A^{(i)} = f_B^{(i)} \cdot h^{xy \cdot k_i}$, for each component k_i of the decoding key. The parties can then compute a subtractive sharing of $xy \cdot \mathbf{k}$ using DDLog.

Let the output of the correctness experiment be defined as 1 if $\langle z \rangle_A - \langle z \rangle_B = P(x_1, \ldots, x_m)$ and defined as 0 otherwise. We will prove that this output is 1 with probability $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$.

We first use a simple hybrid argument to replace the pseudorandom outputs of the PRFs with uniformly random values.

– *Hybrid* $\mathcal{H}_0$. This hybrid is the output of the experiment, as defined above.

– *Hybrid* $\mathcal{H}_1$. This hybrid is identical to the previous hybrid, except that $\langle z_i \rangle_\sigma$ in DEval is computed as $\langle z_i \rangle_\sigma := \mathsf{DDLog}(f_\sigma^{(i)}; r_\sigma^{(i)}) + \hat{r}_\sigma^{(i)} \bmod t$, where $r_\sigma^{(i)} \in \{0,1\}^\lambda$ and $\hat{r}_\sigma^{(i)} \in \mathbb{Z}_t$ are the outputs of truly random functions evaluated at $\mathsf{id} \| i$.

  Claim. $\mathcal{H}_0 \overset{c}{\approx} \mathcal{H}_1$.

  Proof. The claim follows by the pseudorandomness of the PRFs F_1 and F_2. □

Now that we have uniformly random shares, we will prove that the experiment's output is 1, except with a probability of at most $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$. To do so, we first show that if the input for a multiplication satisfies the invariant, the invariant will also hold for the product with probability at least $1 - \varepsilon - \mathsf{negl}(\lambda)$. Then, we use this to derive a lower bound on the probability that the output of the experiment is 1 in hybrid $\mathcal{H}_1$ above.

Claim. For each multiplication instruction $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$ evaluated in DEval, we have

$$\Pr[\langle\!\langle xy \rangle\!\rangle_A - \langle\!\langle xy \rangle\!\rangle_B \neq xy \cdot \mathbf{k} \mid \langle\!\langle y \rangle\!\rangle_A - \langle\!\langle y \rangle\!\rangle_B = y \cdot \mathbf{k}] \leq \varepsilon \cdot \ell + \mathsf{negl}(\lambda).$$

Proof. Consider any arbitrary $i \in [\ell]$. Since $\{\!\{x\}\!\} = (\mathbf{c}_1, \dots, \mathbf{c}_\ell)$ is exponent-linear decodable under $\mathbf{k} = (k_1, \dots, k_\ell)$, we have

$$h^{xy \cdot k_i} = \langle \mathbf{c}_i, y \cdot \mathbf{k} \rangle = \langle \mathbf{c}_i, \langle\!\langle y \rangle\!\rangle_A - \langle\!\langle y \rangle\!\rangle_B \rangle = f_A^{(i)} \cdot \left(f_B^{(i)} \right)^{-1}$$
$$\implies f_A^{(i)} = h^{xy \cdot k_i} \cdot f_B^{(i)},$$

where the second equality follows from the fact that $\langle\!\langle y \rangle\!\rangle_A - \langle\!\langle y \rangle\!\rangle_B = y \cdot \mathbf{k}$.

Let $\langle z_i' \rangle_\sigma = \mathsf{DDLog}(f_\sigma^{(i)}; r_\sigma^{(i)})$, where $r_\sigma^{(i)}$ is the output of a truly random function. Since P is B-bounded and $P(x_1, \dots, x_m) \neq \perp$, we have $|xy| \leq B$. Along with the fact that $|k_i| \leq B_{\mathsf{sk}}$ and $B \cdot B_{\mathsf{sk}} \leq B_{\mathsf{dl}}$, it follows from the correctness of DDLog that $\langle z_i' \rangle_A - \langle z_i' \rangle_B \equiv xy \cdot k_i \bmod t$ with a probability of at least $1 - \varepsilon - \mathsf{negl}(\lambda)$. Moreover, since $\langle z_i \rangle_\sigma = \langle z_i' \rangle_\sigma + \hat{r}_\sigma^{(i)} \pmod t$, where $\hat{r}_\sigma^{(i)} \in \mathbb{Z}_t$, we have $\langle z_i \rangle_A - \langle z_i \rangle_B \equiv xy \cdot k_i \bmod t$, except with a probability of at most $\varepsilon + \mathsf{negl}(\lambda)$.

Conditioned on the event that there was no error in DDLog, $\langle z_i \rangle_A$ and $\langle z_i \rangle_B$ are uniformly random subtractive shares over $\mathbb{Z}_t$ since $\hat{r}_\sigma^{(i)}$ is uniformly random in $\mathbb{Z}_t$. This implies that $\langle z_i \rangle_A - \langle z_i \rangle_B$ does not wrap around t with overwhelming probability.

In more detail, since $|xy \cdot k_i| < B \cdot B_{\mathsf{sk}}$, $\langle z_i \rangle_A - \langle z_i \rangle_B$ wraps around t only when $-t/2 \leq \langle z_i \rangle_B \leq -t/2 + B \cdot B_{\mathsf{sk}}$ or $t/2 - B \cdot B_{\mathsf{sk}} \leq \langle z_i \rangle_B \leq t/2$. The size of this interval is $2B \cdot B_{\mathsf{sk}}$ and since $\langle z_i \rangle_A$ is a uniformly random subtractive share over $\mathbb{Z}_t$, the probability that $\langle z_i \rangle_A - \langle z_i \rangle_B$ wraps around is at most $2B \cdot B_{\mathsf{sk}}/t < 2 \cdot 2^{-\lambda}$, which is negligible. Thus, it follows that $(\langle z_i \rangle_A, \langle z_i \rangle_B)$ constitute a subtractive sharing of $xy \cdot k_i$ over the integers except with a probability of at most $\varepsilon + \mathsf{negl}(\lambda)$, where the probability is over the correctness of DDLog as well as the possibility of wrap-around when converting additive shares over $\mathbb{Z}_t$ to subtractive shares over integers.

Finally, observe that $\langle\!\langle xy \rangle\!\rangle_A - \langle\!\langle xy \rangle\!\rangle_B = xy \cdot \mathbf{k}$ if and only if $\langle z_i \rangle_A - \langle z_i \rangle_B = xy \cdot k_i$, for every $i \in [\ell]$. Therefore, each $(\langle z_i \rangle_A, \langle z_i \rangle_B)$ constitutes a subtractive sharing of $xy \cdot k_i$, except with a probability of at most $\varepsilon + \mathsf{negl}(\lambda)$, since they are computed with independently sampled randomness. By a union bound, and the fact that $\ell \leq \mathsf{poly}(\lambda)$, we have

$$\Pr[\langle\!\langle xy \rangle\!\rangle_A - \langle\!\langle xy \rangle\!\rangle_B \neq xy \cdot \mathbf{k} \mid \langle\!\langle y \rangle\!\rangle_A - \langle\!\langle y \rangle\!\rangle_B = y \cdot \mathbf{k}] \leq \varepsilon \cdot \ell + \mathsf{negl}(\lambda).$$

□

We will argue that if the invariant is true for memory values corresponding to the output of the first $\mathsf{id} - 1$ instructions of the RMS program P, then the invariant is true for the memory value that corresponds to the output of the id-th instruction, except with a probability of at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$. In more detail, observe that if the id-th instruction is an addition instruction $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y)$, then it follows from the additive homomorphism of subtractive sharing that the invariant holds for M_{x+y} with probability 1 since $\langle\!\langle x+y \rangle\!\rangle_A - \langle\!\langle x+y \rangle\!\rangle_B = \langle\!\langle x \rangle\!\rangle_A + \langle\!\langle y \rangle\!\rangle_A - \langle\!\langle x \rangle\!\rangle_B - \langle\!\langle y \rangle\!\rangle_B = x + y$. Similarly, if the id-th instruction is a multiplication instruction $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$, then it follows from our previous claim that the invariant holds for M_{xy} except with a probability of at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$. Finally, if the id-th instruction is a $\mathsf{Convert}(\mathsf{I}_x)$ instruction, then DEval runs the same steps as for evaluating $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_1)$, where $\langle\!\langle 1 \rangle\!\rangle_\sigma = \mathbf{k}_\sigma$. Observe that $(\mathbf{k}_A, \mathbf{k}_B)$ is, by definition, a subtractive sharing of $1 \cdot \mathbf{k}$, which implies from our previous claim that $(\langle\!\langle x \rangle\!\rangle_A, \langle\!\langle x \rangle\!\rangle_B)$ constitutes a subtractive sharing of $x \cdot \mathbf{k}$, except with a probability of at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$. Thus, the invariant holds true for the output M_x of the $\mathsf{Convert}(\mathsf{I}_x)$ instruction.

Since the last component of the decoding key $k_\ell = 1$, we have $\langle z \rangle_A - \langle z \rangle_B = P(x_1, \ldots, x_m)$ in this hybrid when the invariant is true for the memory value M_z corresponding to the output instruction $\mathsf{Output}(\mathsf{M}_z)$. The probability that the invariant does not hold for a memory value is at most $\varepsilon \cdot \ell + \mathsf{negl}(\lambda)$, since the randomness is freshly sampled for each instruction. It thus follows from a straightforward union bound and the fact that $|P| \leq \mathsf{poly}(\lambda)$ that $\langle z \rangle_A - \langle z \rangle_B = P(x_1, \ldots, x_m)$ and the output of the experiment is 1 in hybrid $\mathcal{H}_1$, except with a probability of at most $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$. Moreover, since $\mathcal{H}_0 \overset{c}{\approx} \mathcal{H}_1$, it follows that the output of the experiment is 1 in hybrid $\mathcal{H}_0$, except with a probability of at most $\varepsilon \cdot \ell \cdot |P| + \mathsf{negl}(\lambda)$. This concludes the proof. ■

4 Multi-Key Homomorphic Secret Sharing

In this section, we first formalize the notion of multi-key HSS (MKHSS) in Section 4.1. We then instantiate MKHSS from the NIDLS framework in Section 4.3 and from DDH in Section 4.4.

4.1 Definition

We define multi-key HSS (MKHSS) in Definition 6. An MKHSS scheme allows a party, given a common reference string, to locally generate a key pair and share its input using its public key. These shares can then be used with the input shares computed by any other party (generated using its own public key), to compute subtractive shares of a program's output, evaluated on the joint inputs. This ability to compute shares of the input, independent of the other party's key—indeed, even before knowing the identity of the other party—is the key property of MKHSS schemes.

Let $[\![x]\!]_A^A$ denote Alice's share of her input x and let $[\![x]\!]_B^A$ denote the share of x intended for other parties. The security of the scheme requires that $[\![x]\!]_B^A$—which can be viewed as a ciphertext that enables computing on x—preserves privacy of Alice's input. In contrast, the definition does not impose any security requirements on Alice's share $[\![x]\!]_A^A$, since she already knows the input x as well as the secret key corresponding to the public key used to generate the shares.

We first introduce some additional notation and then proceed with the definition.

Notation. We denote by $[\![x]\!]^\sigma = ([\![x]\!]_A^\sigma, [\![x]\!]_B^\sigma)$ an input sharing of a message x generated using party σ's HSS public key. Additionally, we occasionally write $[\![\mathbf{x}]\!]^\sigma = ([\![x_1]\!]^\sigma, \ldots, [\![x_\ell]\!]^\sigma)$ to denote a *tuple* of input shares of $\mathbf{x} \in \mathcal{R}^\ell$, where $\mathbf{x} = (x_1, \ldots, x_\ell)$. For some party identifier $\sigma \in \{A, B\}$, we write $1 - \sigma$ as shorthand for the "other party identifier" $\overline{\sigma} \in \{A, B\} \setminus \{\sigma\}$.

Definition 6 (Multi-Key Homomorphic Secret Sharing)**.** *A multi-key homomorphic secret sharing (MKHSS) scheme for a program class $\mathcal{P}$, defined over a ring $\mathcal{R}$, and having a message space $\mathcal{M} \subseteq \mathcal{R}$ consists of four efficient algorithms* $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ *with the following syntax:*

– $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs.

– $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$. *The randomized key generation algorithm, independently run by each party, takes as input the CRS* crs *and outputs a public and private key pair* $(\mathsf{pk}, \mathsf{sk})$.

– $\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x) \to ([\![x]\!]_A^\sigma, [\![x]\!]_B^\sigma)$. *The randomized share algorithm takes as input the CRS* crs*, the party identifier $\sigma \in \{A, B\}$, the party's public key* pk_σ*, and a message $x \in \mathcal{M}$. It outputs a pair of input shares $([\![x]\!]_A^\sigma, [\![x]\!]_B^\sigma)$ encoding the message x.*

– $\mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]_\sigma^A, [\![\mathbf{x}_B]\!]_\sigma^B, P) \to \langle z \rangle_\sigma^\mathcal{R}$. *The deterministic evaluation algorithm takes as input the CRS* crs*, the party identifier $\sigma \in \{A, B\}$, the party's secret key* sk_σ*, the public key of another party* $\mathsf{pk}_{1-\sigma}$*, two tuples $[\![\mathbf{x}_A]\!]_\sigma^A$ and $[\![\mathbf{x}_B]\!]_\sigma^B$ of the party's input shares (where the tuples are generated by different parties using* Share*), and a program description P. It outputs a subtractive share (over the ring $\mathcal{R}$) of the evaluation result z.*

An MKHSS scheme must satisfy the following correctness and security properties.

*Correctness. An MKHSS scheme is said to be ε-*correct*, for some $\varepsilon \in [0,1)$, if for all $\lambda \in \mathbb{N}$, all $2m$-input programs $P \in \mathcal{P}$, and all $\mathbf{x}_A, \mathbf{x}_B \in \mathcal{M}^m$, we have*

$$\Pr\left[\begin{array}{c} \langle z\rangle_A^{\mathcal{R}} - \langle z\rangle_B^{\mathcal{R}} \\ \neq \\ P(\mathbf{x}_A, \mathbf{x}_B) \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs}),\ \forall \sigma \in \{A,B\} \\ ([\![\mathbf{x}_\sigma]\!]_A^\sigma, [\![\mathbf{x}_\sigma]\!]_B^\sigma) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma),\ \forall \sigma \in \{A,B\} \\ \langle z\rangle_\sigma^{\mathcal{R}} \leftarrow \mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]_\sigma^A, [\![\mathbf{x}_B]\!]_\sigma^B, P),\ \forall \sigma \in \{A,B\} \end{array}\right] \leq \varepsilon + \mathsf{negl}(\lambda),$$

where $\mathbf{x}_\sigma = (x_\sigma^{(1)}, \ldots, x_\sigma^{(m)})$ for each $\sigma \in \{A,B\}$. Note that we slightly abuse notation by letting $\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma)$ *denote running* $\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x_\sigma^{(i)})$ *separately for each i. If $\varepsilon = 0$, we simply say that the MKHSS is* correct.

Security. An MKHSS scheme is said to be secure *if for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, and all $\sigma \in \{A,B\}$, we have that*

$$\Pr\left[b' = b \;:\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_\sigma) \\ b \leftarrow\!\$\, \{0,1\} \\ ([\![x_b]\!]_A^\sigma, [\![x_b]\!]_B^\sigma) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x_b) \\ b' \leftarrow \mathcal{A}\left([\![x_b]\!]_{1-\sigma}^\sigma, \mathsf{st}\right) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Comparison to public-key HSS. In a public-key HSS scheme (e.g., [BGI16, Definition 2.2]), the Setup algorithm outputs a public key pk and two private evaluations keys ek_A and ek_B. Any party—not necessarily those holding the evaluation keys—can then share their input using pk, which in turn allows the servers holding the evaluation keys to non-interactively compute on all shared inputs. Thus, compared to an MKHSS scheme, a public-key HSS scheme allows computing on the inputs of several parties; but this comes at the cost of requiring a correlated setup or, alternatively, a PKI [BGI17, OSY21, ADOS22], which implies a two-round sharing protocol in the CRS model.

While an MKHSS scheme and a public-key HSS scheme might initially seem incomparable, it is not too hard to see that the former implies the latter. Specifically, given an MKHSS scheme MKHSS, a public-key HSS scheme can be constructed as follows.

– The setup algorithm computes crs using $\mathsf{MKHSS.Setup}$, generates two keys pairs $(\mathsf{pk}_A, \mathsf{sk}_A)$ and $(\mathsf{pk}_B, \mathsf{sk}_B)$ using $\mathsf{MKHSS.KeyGen}$ and outputs $\mathsf{pk} := (\mathsf{crs}, \mathsf{pk}_A, \mathsf{pk}_B)$, $\mathsf{ek}_A := \mathsf{sk}_A$, and $\mathsf{ek}_B := \mathsf{sk}_B$.
– The HSS share of an input x is computed using pk by first computing a subtractive sharing, $\langle x\rangle = (\langle x\rangle_A, \langle x\rangle_B)$, and then computing MKHSS shares of $\langle x\rangle_\sigma$ using pk_σ i.e.,

$$\left([\![\langle x\rangle_A]\!]_A^A, [\![\langle x\rangle_A]\!]_B^A\right) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, A, \mathsf{pk}_A, x)$$
$$\left([\![\langle x\rangle_B]\!]_A^B, [\![\langle x\rangle_B]\!]_B^B\right) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, B, \mathsf{pk}_B, x).$$

Thus, $([\![\langle x\rangle_A]\!]_A^A, [\![\langle x\rangle_B]\!]_A^B)$ constitutes the HSS share of x for the server holding ek_A and $([\![\langle x\rangle_A]\!]_B^A, [\![\langle x\rangle_B]\!]_B^B)$ is the HSS share for the server holding ek_B. In particular, although party-σ might learn $\langle x\rangle_\sigma$, the security of MKHSS ensures the privacy of $\langle x\rangle_{1-\sigma}$, thereby preserving the privacy of x.
– To evaluate a program P on the shared inputs, the servers use $\mathsf{MKHSS.Eval}$ to evaluate a program P' that first reconstructs the inputs and then evaluates P.

This gives a public-key HSS scheme for all programs P for which the corresponding program P' can be evaluated using the MKHSS scheme. Specifically, for MKHSS schemes for polynomial-size RMS programs—which are the focus of this work—this implies a public-key HSS scheme for polynomial-size RMS programs.

4.2 External security

We introduce an additional security notion for MKHSS, which will be important for our applications. The notion strengthens the correctness property of MKHSS by requiring, informally, that the output

```
$\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,0}(\lambda)$:
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
foreach $\sigma \in \{A, B\}$:
    $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$
    $([\![\mathbf{x}_\sigma]\!]^\sigma_A, [\![\mathbf{x}_\sigma]\!]^\sigma_B) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma)$
    $\langle z \rangle^{\mathcal{R}}_\sigma := \mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]^A_\sigma, [\![\mathbf{x}_B]\!]^B_\sigma, P)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_A, \mathsf{pk}_B, [\![\mathbf{x}_A]\!]^A_B, [\![\mathbf{x}_B]\!]^B_A, \langle z \rangle^{\mathcal{R}}_A, \langle z \rangle^{\mathcal{R}}_B)$
return $b$
```

```
$\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,1}(\lambda)$:
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
foreach $\sigma \in \{A, B\}$:
    $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$
    $([\![\mathbf{x}_\sigma]\!]^\sigma_A, [\![\mathbf{x}_\sigma]\!]^\sigma_B) \leftarrow \mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathbf{x}_\sigma)$
$\langle z \rangle^{\mathcal{R}}_B \leftarrow\$ \mathcal{R}$
$\langle z \rangle^{\mathcal{R}}_A := \langle z \rangle^{\mathcal{R}}_B + P(\mathbf{x}_A, \mathbf{x}_B)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_A, \mathsf{pk}_B, [\![\mathbf{x}_A]\!]^A_B, [\![\mathbf{x}_B]\!]^B_A, \langle z \rangle^{\mathcal{R}}_A, \langle z \rangle^{\mathcal{R}}_B)$
return $b$
```

Fig. 2: External security experiment for MKHSS.

shares of any HSS evaluation are indistinguishable from uniformly random subtractive shares of the output over the ring $\mathcal{R}$.

Definition 7 (External Security of Multi-Key Homomorphic Secret Sharing)**.** *An MKHSS scheme* $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ *for a program class $\mathcal{P}$, defined over a ring $\mathcal{R}$, is* externally secure *if for all $\lambda \in \mathbb{N}$, all $2m$-input programs $P \in \mathcal{P}$, all $\mathbf{x}_A, \mathbf{x}_B \in \mathcal{M}^m$, and all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\mathsf{Adv}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B}(\lambda) := \left|\Pr\left[\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,A}(\lambda) = 1\right] - \Pr\left[\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,B}(\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{E}^{\mathsf{exsec}}_{\mathcal{A},\mathbf{x}_A,\mathbf{x}_B,b}(\lambda)$ is defined in Figure 2.

Getting external security, generically. We now show a simple transformation for converting any MKHSS scheme $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ into an MKHSS scheme MKHSS^* that satisfies external security. The idea is to use a non-interactive key exchange (NIKE) to derive a common pseudorandom key, which we use to randomize the output shares. Let $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ be a NIKE scheme (cf. Definition 1) and let G be a PRG (note that MKHSS implies the existence of NIKE, generically [BGI+18]). We describe the transformation to external security in Figure 3; the main idea is to have $\mathsf{MKHSS}^*.\mathsf{Eval}$ derive a common pseudorandom string K which is then used to randomize the output with the help of the PRG.

Claim. The MKHSS scheme described in Figure 3 satisfies Definition 7 (external security).

Proof (sketch). The proof of external security of MKHSS^* is almost immediate and can be shown with a simple hybrid argument. First, invoke the security of NIKE to replace K with a fresh random string. Second, invoke the PRG security to replace $G(K)$ with a fresh random value R. Finally, invoke the correctness of MKHSS to conclude that $\langle z \rangle^{\mathcal{R}}_A + R$, $\langle z \rangle^{\mathcal{R}}_B + R$ are distributed as pseudorandom subtractive shares of $P(\mathbf{x}_A, \mathbf{x}_B)$ over the ring $\mathcal{R}$. ■

4.3 MKHSS in the NIDLS framework

In this section, we construct multi-key HSS in the NIDLS framework (cf. Definition 5). We first recall the relevant assumptions in the NIDLS framework and the "NIDLS ElGamal" encryption scheme from Abram et al. [ADOS22]. The encryption scheme instantiates ElGamal over a NIDLS group, and helps simplify the presentation of our MKHSS construction.

Assumptions. Our construction requires the same assumptions as the HSS scheme presented in [ADOS22], namely, the Decisional Diffie–Hellman (DDH) assumption and the small exponent assumption over the NIDLS group.

Definition 8 (NIDLS Decisional Diffie–Hellman)**.** *The Decisional Diffie–Hellman (DDH) assumption is said to hold in the NIDLS framework if for every efficient adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$ we have*

$$|\Pr[\mathcal{A}(\mathsf{par}, \rho, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(\mathsf{par}, \rho, g, g^x, g^y, g^z) = 1]| \leq \mathsf{negl}(\lambda),$$

External-Security Transformation for MKHSS

Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be any MKHSS scheme, let $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyDer})$ be any NIKE scheme, and let $G : \{0,1\}^\lambda \to \mathcal{R}$ be a PRG.

```
$\mathsf{MKHSS}^*.\mathsf{Setup}(1^\lambda)$:
  1 : $\mathsf{crs}_{\mathsf{mkhss}} \leftarrow \mathsf{MKHSS}.\mathsf{Setup}(1^\lambda)$
  2 : $\mathsf{crs}_{\mathsf{nike}} \leftarrow \mathsf{NIKE}.\mathsf{Setup}(1^\lambda)$
  3 : $\mathsf{crs} := (\mathsf{crs}_{\mathsf{mkhss}}, \mathsf{crs}_{\mathsf{nike}})$
  4 : return $\mathsf{crs}$
```

```
$\mathsf{MKHSS}^*.\mathsf{KeyGen}(\mathsf{crs})$:
  1 : parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{mkhss}}, \mathsf{crs}_{\mathsf{nike}})$
  2 : $(\mathsf{pk}^{\mathsf{nike}}, \mathsf{sk}^{\mathsf{nike}}) \leftarrow \mathsf{NIKE}.\mathsf{KeyGen}(\mathsf{crs}_{\mathsf{nike}})$
  3 : $(\mathsf{pk}^{\mathsf{mkhss}}, \mathsf{sk}^{\mathsf{mkhss}}) \leftarrow \mathsf{MKHSS}.\mathsf{KeyGen}(\mathsf{crs}_{\mathsf{mkhss}})$
  4 : $\mathsf{pk}^* := (\mathsf{pk}^{\mathsf{nike}}, \mathsf{pk})$
  5 : $\mathsf{sk}^* := (\mathsf{sk}^{\mathsf{nike}}, \mathsf{sk})$
  6 : return $(\mathsf{pk}^*, \mathsf{sk}^*)$
```

```
$\mathsf{MKHSS}^*.\mathsf{Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$:
  1 : parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{mkhss}}, \_)$
  2 : $([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B) \leftarrow \mathsf{Share}(\mathsf{crs}_{\mathsf{mkhss}}, \sigma, \mathsf{pk}_\sigma, x)$
  3 : return $([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B)$
```

```
$\mathsf{MKHSS}^*.\mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}^*_\sigma, \mathsf{pk}^*_{1-\sigma}, [\![\mathbf{x}_A]\!]^A_\sigma, [\![\mathbf{x}_B]\!]^B_\sigma, P)$:
  1 : parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{mkhss}}, \mathsf{crs}_{\mathsf{nike}})$
  2 : parse $\mathsf{pk}^*_{1-\sigma} = (\mathsf{pk}^{\mathsf{nike}}_{1-\sigma}, \mathsf{pk}^{\mathsf{mkhss}}_{1-\sigma})$
  3 : parse $\mathsf{sk}^*_\sigma = (\mathsf{sk}^{\mathsf{nike}}_\sigma, \mathsf{sk}^{\mathsf{mkhss}}_\sigma)$
  4 : $\langle z \rangle^{\mathcal{R}}_\sigma := \mathsf{Eval}(\mathsf{crs}, \sigma, \mathsf{sk}^{\mathsf{mkhss}}_\sigma, \mathsf{pk}^{\mathsf{mkhss}}_{1-\sigma}, [\![\mathbf{x}_A]\!]^A_\sigma, [\![\mathbf{x}_B]\!]^B_\sigma, P)$
  5 : $K := \mathsf{KeyDer}(\mathsf{pk}^{\mathsf{nike}}_{1-\sigma}, \mathsf{sk}^{\mathsf{nike}}_\sigma)$
  6 : return $\langle z \rangle^{\mathcal{R}}_\sigma + G(K)$
```

Fig. 3: External-security transformation for MKHSS.

where the probabilities are over the choice of $\mathsf{par} := (\mathbb{G}, \mathbb{H}, \mathbb{K}, h, t, B_{\mathsf{nidls}}, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda)$, $(g, \rho) \leftarrow \mathcal{D}_{\mathsf{nidls}}(1^\lambda, \mathsf{par})$, *and* $x, y, z \leftarrow\$ [B_{\mathsf{nidls}}]$.

Definition 9 (NIDLS Small Exponent Assumption)**.** *The small exponent assumption with length* B_{sk} *is said to hold in the NIDLS framework if for every efficient adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *we have*

$$|\Pr[\mathcal{A}(\mathsf{par}, B_{\mathsf{sk}}, \rho, g, g^x) = 1] - \Pr[\mathcal{A}(\mathsf{par}, B_{\mathsf{sk}}, \rho, g, g^y) = 1]| \le \mathsf{negl}(\lambda),$$

where the probabilities are over the choice of $\mathsf{par} := (\mathbb{G}, \mathbb{H}, \mathbb{K}, h, t, B_{\mathsf{nidls}}, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda)$, $(g, \rho) \leftarrow \mathcal{D}_{\mathsf{nidls}}(1^\lambda, \mathsf{par})$, $x \leftarrow\$ [B_{\mathsf{nidls}}]$, *and* $y \leftarrow\$ [B_{\mathsf{sk}}]$.

NIDLS ElGamal encryption. We recall the details of the ElGamal encryption scheme in the NIDLS framework [ADOS22] in Figure 4. Note that in addition to the standard encryption algorithm, the scheme also includes a "flipped" ElGamal encryption. As discussed in Section 2, the flipped encryption of a message x is a ciphertext ct that decrypts to $\mathsf{sk} \cdot x \bmod t$, where sk is the secret key and t is the order of the NIDLS subgroup $\mathbb{H}$. The encryption scheme is IND-CPA secure assuming the hardness of DDH in the NIDLS framework [ADOS22]. However, we note that in the proof of security of our MKHSS construction, we directly reduce to the NIDLS DDH assumption instead of IND-CPA security of the encryption scheme to simplify the presentation.

NIDLS MKHSS construction. We present our MKHSS construction in the NIDLS framework in Figure 5. The construction closely follows the scheme presented in the technical overview (cf. Section 2). In addition to the MKHSS algorithms, we define two subprocedures that capture synchronization of input shares. The procedure $\mathsf{ExpLinEncS}$ allows the party sharing the input to compute a synchronized input share under the concatenated secret key while $\mathsf{ExpLinEncR}$ allows the other party,

NIDLS ElGamal Encryption [ADOS22]

Public Parameters. Algorithms GGen and $\mathcal{D}_{\mathsf{nidls}}$ that realize the NIDLS framework (cf. Definition 5).

```
EG.Setup(1^λ):
  1 : $\mathsf{par} \leftarrow \mathsf{GGen}(1^\lambda)$
  2 : $(g, \rho) \leftarrow\$ \mathcal{D}_{\mathsf{nidls}}(1^\lambda, \mathsf{par})$
  3 : $\mathsf{crs} := (\mathsf{par}, g, \rho)$
  4 : return crs

EG.KeyGen(crs):
  1 : parse $B_{\mathsf{nidls}}, g$ from crs
  2 : $s \leftarrow\$ [B_{\mathsf{nidls}}]$
  3 : $f := g^{-s}$
  4 : $(\mathsf{pk}, \mathsf{sk}) := (f, s)$
  5 : return (pk, sk)

EG.Encrypt(crs, pk, x):
  1 : parse $h, B_{\mathsf{nidls}}, g$ from crs
      parse pk = $f$
  2 : $r \leftarrow\$ [B_{\mathsf{nidls}}]$
  3 : $\mathsf{ct} := (g^r, f^r \cdot h^x)$
  4 : return ct

EG.FlipEncrypt(crs, pk, x):
  1 : parse $h, B_{\mathsf{nidls}}, g$ from crs
      parse pk = $f$
  2 : $r \leftarrow\$ [B_{\mathsf{nidls}}]$
  3 : $\mathsf{ct} := (g^r \cdot h^x, f^r)$
  4 : return ct

EG.Decrypt(sk, ct):
  1 : parse ct = $(c_1, c_2)$
  2 : $x := \log_h(c_1^{\mathsf{sk}} \cdot c_2)$
  3 : return $x$
```

Fig. 4: ElGamal encryption in the NIDLS framework.

receiving the input share, to synchronize. As discussed in Section 2, a key property of our construction is that both parties obtain identical, “exponent-linear decodable” shares upon synchronization.

Performance analysis. The share of each input sent to the other party consists of six group elements and synchronizing each input share requires at most four group exponentiations. Note that, when evaluating each multiplication instruction in DEval (cf. Figure 1), it suffices to use one among $\mathbf{c}_2^\sigma$ and $\mathbf{c}_2^{1-\sigma}$ since the corresponding components in the concatenated secret key $(\mathsf{sk}_A, 1, \mathsf{sk}_B, 1)$ are equal. Furthermore, since two among the four group elements in $\mathbf{c}_1^{1-\sigma}$ and $\mathbf{c}_2^\sigma$ are the identity, evaluating each multiplication instruction requires eight group exponentiations and four distributed discrete logarithm computations. Compared to the NIDLS-based HSS construction of Abram et al. [ADOS22], the MKHSS construction requires communicating two additional group elements per input and has a computational overhead of 2×. When instantiated over the Paillier group with a 3072-bit modulus N, where each group exponentiation takes approximately 15 milliseconds, this results in a per-multiplication cost of around 120 to 150 milliseconds. Thus, the MKHSS construction is potentially practical when implemented, albeit an order of magnitude slower when compared to efficient non-multi-key HSS schemes [BCG+17].

Theorem 1. *Let $t = t(\lambda)$ be the order of the subgroup in the NIDLS framework (cf. Definition 5). If the DDH assumption and the small exponent assumption with length B_{sk} hold in the NIDLS framework, then the construction described in Figure 5 is an MKHSS scheme for the class of polynomial sized RMS programs with bound B and message space $\mathbb{Z}_B$ where $B < t/(B_{\mathsf{sk}} \cdot 2^\lambda)$ and λ is the security parameter of the MKHSS scheme.*

Proof. We first show that the construction satisfies the correctness property and then proceed to argue its security.

Correctness. Recall that the correctness property requires that parties obtain a subtractive sharing of the program output upon evaluation.

We first show that for an input x shared by party-σ, where $\sigma \in \{A, B\}$, the parties obtain the same encoding $\{\!\{x\}\!\}$ when party-σ runs $\mathsf{ExpLinEncS}$ on its share $[\![x]\!]_\sigma^\sigma$ and party-$(1-\sigma)$ runs $\mathsf{ExpLinEncR}$ on its share $[\![x]\!]_{1-\sigma}^\sigma$. Moreover, we prove that $\{\!\{x\}\!\}$ is exponent-linear decodable under the decoding key $\mathbf{k} = (s_A, 1, s_B, 1)$.

NIDLS-based MKHSS

Public Parameters. Let $(\mathsf{GGen}, \mathcal{D}_{\mathsf{nidls}}, \mathsf{DDLog})$ be the algorithms provided by the NIDLS framework (cf. Definition 5), let B_{sk} be a bound for the small exponent assumption. We use the NIDLS ElGamal Encryption (cf. Figure 4) instantiated using GGen and $\mathcal{D}_{\mathsf{nidls}}$. Finally, we use DEval (cf. Figure 1) instantiated using DDLog, and the subroutines $\mathsf{ExpLinEncS}$ and $\mathsf{ExpLinEncR}$ defined in Figure 6.

Notation. Let $\mathbf{1}_{\mathbb{G}} = (1_{\mathbb{G}}, 1_{\mathbb{G}})$ where $1_{\mathbb{G}}$ is the identity element in $\mathbb{G}$.

```
$\mathsf{MKHSS.Setup}(1^\lambda)$:
    $\mathsf{crs_{enc}} \leftarrow \mathsf{EG.Setup}(1^\lambda)$
    $k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}} \leftarrow\$ \{0,1\}^\lambda$
    $\mathsf{crs} := (\mathsf{crs_{enc}}, k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}})$
    **return** $\mathsf{crs}$
```

```
$\mathsf{MKHSS.KeyGen}(\mathsf{crs})$:
    **parse** $g$ **from** $\mathsf{crs}$
    $s \leftarrow\$ [B_{\mathsf{sk}}],\ f := g^{-s}$
    $(\mathsf{epk}, \mathsf{esk}) := (f, s)$
    $\mathsf{pk} := (\mathsf{crs}, \mathsf{epk})$
    $\mathsf{sk} := (\mathsf{pk}, \mathsf{esk})$
    **return** $(\mathsf{pk}, \mathsf{sk})$
```

```
$\mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$:
    **parse** $\mathsf{crs_{enc}}$ **from** $\mathsf{crs}$
    **parse** $h, B_{\mathsf{nidls}}, g$ **from** $\mathsf{crs_{enc}}$
    $r_1, u_1 \leftarrow [B_{\mathsf{nidls}}]$
    $\mathsf{ct}_1 := \mathsf{EG.FlipEncrypt}(\mathsf{crs_{enc}}, \mathsf{epk}_\sigma, x; r_1)$
    $\mathsf{rct}_1 := (g^{u_1}, f_\sigma^{u_1} \cdot g^{r_1})$
    $\mathsf{ct}_2 := \mathsf{EG.Encrypt}(\mathsf{crs_{enc}}, \mathsf{epk}_\sigma, x)$
    $[\![x]\!]_\sigma^\sigma := (\mathsf{ct}_1, \mathsf{ct}_2, r_1, u_1)$
    $[\![x]\!]_{1-\sigma}^\sigma := (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rct}_1)$
    **return** $([\![x]\!]_A^\sigma, [\![x]\!]_B^\sigma)$
```

```
$\mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_0]\!]_\sigma^0, [\![\mathbf{x}_1]\!]_\sigma^1, P)$:
    **parse** $s_\sigma$ **from** $\mathsf{sk}_\sigma$ **and** $k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}$ **from** $\mathsf{crs}$
    **parse** $[\![\mathbf{x}_A]\!]_\sigma^A = \left([\![x_A^{(1)}]\!]_\sigma^A, \ldots, [\![x_A^{(m)}]\!]_\sigma^A\right)$
    **parse** $[\![\mathbf{x}_B]\!]_\sigma^B = \left([\![x_B^{(1)}]\!]_\sigma^B, \ldots, [\![x_B^{(m)}]\!]_\sigma^B\right)$
    **for** $i \in [m]$
        $\{\!\{x_\sigma^{(i)}\}\!\} := \mathsf{ExpLinEncS}\left(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x_\sigma^{(i)}]\!]_\sigma^\sigma\right)$
        $\{\!\{x_{1-\sigma}^{(i)}\}\!\} := \mathsf{ExpLinEncR}\left(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x_{1-\sigma}^{(i)}]\!]_\sigma^{1-\sigma}\right)$
    $\{\!\{\mathbf{x}\}\!\} := \left(\{\!\{x_A^{(1)}\}\!\}, \ldots, \{\!\{x_A^{(m)}\}\!\}, \{\!\{x_B^{(1)}\}\!\}, \ldots, \{\!\{x_B^{(m)}\}\!\}\right)$
    $\mathbf{k}_\sigma := (s_\sigma, 1, 0, 0)$ **if** $\sigma = A$ **else** $(0, 0, -s_\sigma, -1)$
    $\mathsf{ek}_\sigma := (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \mathbf{k}_\sigma)$
    **return** $\mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, \{\!\{\mathbf{x}\}\!\}, P)$
```

Fig. 5: MKHSS in the NIDLS framework.

Claim. For all integers $x \in \mathbb{Z}_t$ and all $\sigma \in \{A, B\}$, we have

$$\{\!\{x\}\!\} = \mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]_\sigma^\sigma) = \mathsf{ExpLinEncR}(\mathsf{sk}_{1-\sigma}, \mathsf{pk}_\sigma, [\![x]\!]_{1-\sigma}^\sigma),$$

where $([\![x]\!]_A^\sigma, [\![x]\!]_B^\sigma) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$. Moreover, $\{\!\{x\}\!\}$ is base-h exponent-linear decodable under the decoding key $\mathbf{k} = (s_A, 1, s_B, 1)$.

Proof. We consider the case when $\sigma = A$ for ease of exposition; a similar argument follows for the case when $\sigma = B$. From the description of $\mathsf{MKHSS.Share}$, we have $[\![x]\!]_A^A = (\mathsf{ct}_1, \mathsf{ct}_2, r_1, u_1)$ and $[\![x]\!]_B^A = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rct}_1)$, where

$$\mathsf{ct}_1 = (g^{r_1} \cdot h^x,\ f_A^{r_1}),$$
$$\mathsf{ct}_2 = (g^{r_2},\ f_A^{r_2} \cdot h^x),$$
$$\mathsf{rct}_1 = (g^{u_1},\ f_A^{u_1} \cdot g^{r_1}).$$

Now, observe that party-B computes ct' in $\mathsf{ExpLinEncR}$ as

$$\mathsf{ct}' = \left((g^{u_1})^{-s_B}, (f_A^{u_1} \cdot g^{r_1})^{-s_B} \cdot (f_A^{r_1})^{-1}\right)$$
$$= \left(f_B^{u_1}, f_B^{-s_A \cdot u_1} \cdot f_B^{r_1} \cdot f_A^{-r_1}\right),$$

```
$\mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]^\sigma_\sigma)$:
  1 : parse $[\![x]\!]^\sigma_\sigma = (\mathsf{ct}_1, \mathsf{ct}_2, r_1, u_1)$
  2 : $\mathbf{c}^\sigma_1 := \mathsf{ct}_1 \| \mathbf{1}_\mathbb{G}$ if $\sigma = A$ else $\mathbf{1}_\mathbb{G} \| \mathsf{ct}_1$
  3 : $\mathsf{ct}' := (f^{u_1}_{1-\sigma}, f^{-s_\sigma \cdot u_1}_{1-\sigma} \cdot f^{r_1}_{1-\sigma} \cdot f^{-r_1}_\sigma)$
  4 : $\mathbf{c}^{1-\sigma}_1 := \mathsf{ct}' \| \mathsf{ct}_1$ if $\sigma = A$ else $\mathsf{ct}_1 \| \mathsf{ct}'$
  5 : $\mathbf{c}^\sigma_2 := \mathsf{ct}_2 \| \mathbf{1}_\mathbb{G}$ if $\sigma = A$ else $\mathbf{1}_\mathbb{G} \| \mathsf{ct}_2$
  6 : $\mathbf{c}^{1-\sigma}_2 := \mathbf{c}^\sigma_2$
  7 : $\{\!\{x\}\!\} := \left(\mathbf{c}^A_1, \mathbf{c}^A_2, \mathbf{c}^B_1, \mathbf{c}^B_2\right)$
  8 : return $\{\!\{x\}\!\}$

$\mathsf{ExpLinEncR}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]^{1-\sigma}_\sigma)$:
  1 : parse $[\![x]\!]^{1-\sigma}_\sigma = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rct}_1)$
  2 : parse $\mathsf{rct}_1 = (g^{u_1}, f^{u_1}_{1-\sigma} \cdot g^{r_1})$
  3 : parse $\mathsf{ct}_1 = (\_, f^{r_1}_{1-\sigma})$
  4 : $\mathsf{ct}' := \left((g^{u_1})^{-s_\sigma}, (f^{u_1}_{1-\sigma} \cdot g^{r_1})^{-s_\sigma} \cdot (f^{r_1}_{1-\sigma})^{-1}\right)$
  5 : $\mathbf{c}^\sigma_1 := \mathsf{ct}_1 \| \mathsf{ct}'$ if $\sigma = A$ else $\mathsf{ct}' \| \mathsf{ct}_1$
  6 : $\mathbf{c}^{1-\sigma}_1 := \mathbf{1}_\mathbb{G} \| \mathsf{ct}_1$ if $\sigma = A$ else $\mathsf{ct}_1 \| \mathbf{1}_\mathbb{G}$
  7 : $\mathbf{c}^\sigma_2 := \mathbf{1}_\mathbb{G} \| \mathsf{ct}_2$ if $\sigma = A$ else $\mathsf{ct}_2 \| \mathbf{1}_\mathbb{G}$
  8 : $\mathbf{c}^{1-\sigma}_2 := \mathbf{c}^\sigma_2$
  9 : $\{\!\{x\}\!\} := (\mathbf{c}^A_1, \mathbf{c}^A_2, \mathbf{c}^B_1, \mathbf{c}^B_2)$
 10 : return $\{\!\{x\}\!\}$
```

Fig. 6: Exponent-linear encoding algorithms used as subroutines in the NIDLS MKHSS construction.

which is identical to ct' computed by party-A in $\mathsf{ExpLinEncS}$, where the second equality follows from the fact that $f_A = g^{-s_A}$ and $f_B = g^{-s_B}$. It is then easy to see that both parties obtain $\{\!\{x\}\!\} = (\mathbf{c}^A_1, \mathbf{c}^A_2, \mathbf{c}^B_1, \mathbf{c}^B_2)$ where

$$\begin{aligned}
\mathbf{c}^A_1 &= (g^{r_1} \cdot h^x,\ f^{r_1}_A,\ g^0,\ g^0),\\
\mathbf{c}^A_2 &= (g^{r_2},\ f^{r_2}_A \cdot h^x,\ g^0,\ g^0),\\
\mathbf{c}^B_1 &= (f^{u_1}_B,\ f^{-s_A \cdot u_1}_B \cdot f^{r_1}_B \cdot f^{-r_1}_A,\ g^{r_1} \cdot h^x,\ f^{r_1}_A),\\
\mathbf{c}^B_2 &= (g^{r_2},\ f^{r_2}_A \cdot h^x,\ g^0,\ g^0).
\end{aligned}$$

We are left to show that $\{\!\{x\}\!\}$ is base-h exponent-linear decodable under $\mathbf{k} = (k_1, k_2, k_3, k_4) = (s_A, 1, s_B, 1)$. Observe that ct_1 and ct_2 are encryptions of $x \cdot k_1 = x \cdot s_A$ and $x \cdot k_2 = x$ under the public key f_A since they are computed using $\mathsf{FlipEncrypt}$ and $\mathsf{Encrypt}$ respectively. This implies that $\langle \mathsf{ct}_i, (s_A, 1)\rangle = h^{x \cdot k_i}$ for each $i \in \{1, 2\}$. However, for each $i \in \{1, 2\}$, we have $\mathbf{c}^A_i = \mathsf{ct}_i \| (g^0, g^0)$, which implies that $\langle \mathbf{c}^A_i, \mathbf{k}\rangle = \langle \mathsf{ct}_i, (s_A, 1)\rangle = h^{x \cdot k_i}$. Moreover, since $k_4 = k_2 = 1$ and $\mathbf{c}^B_2 = \mathbf{c}^A_2$, we have $\langle \mathbf{c}^B_2, \mathbf{k}\rangle = \langle \mathbf{c}^A_2, \mathbf{k}\rangle = h^{x \cdot k_2} = h^{x \cdot k_4}$. Finally, we have

$$\begin{aligned}
\langle \mathbf{c}^B_1, \mathbf{k}\rangle &= (f^{u_1}_B)^{s_A} \cdot f^{-s_A \cdot u_1}_B \cdot f^{r_1}_B \cdot f^{-r_1}_A \cdot (g^{r_1} \cdot h^x)^{s_B} \cdot f^{r_1}_A\\
&= f^{r_1}_B \cdot g^{r_1 \cdot s_B} \cdot h^{x \cdot s_B}\\
&= h^{x \cdot s_B},
\end{aligned}$$

where the third equality follows from the fact that $f_B = g^{-s_B}$. In turn, this implies that $g^{r_1 \cdot s_B} = f^{-r_1}_B$. Then, we have that $\langle \mathbf{c}^B_1, \mathbf{k}\rangle = h^{x \cdot k_3}$, which in turn implies that $\{\!\{x\}\!\}$ is indeed a base-h exponent-linear decodable under $\mathbf{k}$. $\square$

To complete the proof of correctness, observe that $\mathbf{k}_A$ and $\mathbf{k}_B$, computed by party-A and party-B in $\mathsf{MKHSS.Eval}$, form a subtractive sharing of $\mathbf{k}$ because $\mathbf{k}_A - \mathbf{k}_B = (s_A, 1, 0, 0) - (0, 0, -s_B, -1) = (s_A, 1, s_B, 1) = \mathbf{k}$.

In sum, it follows that parties run DEval with encodings of the input that are base-h exponent-linear decodable. Furthermore, we have $s_A, s_B \le B_{\mathsf{sk}}$ by definition, which implies that each component of $\mathbf{k}$ is bounded by B_{sk}. Since $B \cdot B_{\mathsf{sk}} \cdot 2^\lambda < t$ and DDLog is a $(B \cdot B_{\mathsf{sk}})$-bounded base-$h$ algorithm for distributed discrete logarithm with negligible correctness error, it follows from Lemma 3 that the MKHSS scheme satisfies the correctness property for all polynomial-size RMS programs P.

Security. The security property requires that the input share $[\![x]\!]^\sigma_{1-\sigma}$ of party-$(1-\sigma)$, ensures the privacy of an input x shared using party-σ's public key pk_σ. Recall that the public key pk_σ consists of $f_\sigma = g^{-s_\sigma}$ while the share $[\![x]\!]^\sigma_{1-\sigma} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rct}_1)$ where

$$\begin{aligned}
\mathsf{ct}_1 &= (g^{r_1} \cdot h^x,\ f^{r_1}_\sigma),\\
\mathsf{ct}_2 &= (g^{r_2},\ f^{r_2}_\sigma \cdot h^x),\\
\mathsf{rct}_1 &= (g^{u_1},\ f^{u_1}_\sigma \cdot g^{r_1}),
\end{aligned}$$

and $r_1, r_2, u_1 \leftarrow\$ [B_{\mathsf{nidls}}]$. At a high level, ct_1 is an encryption of $x \cdot s_\sigma$, ct_2 is an encryption of x and rct_1 is an encryption of the randomness r_1 used for computing ct_1. Moreover, ct_1, ct_2 and rct_1 are all encryptions under the public key f_σ, computed using independently sampled randomness. Thus, security of the MKHSS scheme follows from the indistinguishability of the ciphertexts. However, we note that rct_1 is not a valid ciphertext under the NIDLS ElGamal scheme, since we use g^{r_1} instead of h^{r_1}—i.e., it might not be possible to decrypt rct_1 and obtain r_1 using the secret key s_σ. Nevertheless, this does not pose an issue to argue indistinguishability of rct_1 since under the DDH assumption $f_\sigma^{u_1}$ is indistinguishable from a uniformly random element in the subgroup generated by g.

We now proceed to prove formally prove the above sketch. Consider any efficient adversary $\mathcal{A}$ for the security experiment defined in Definition 6. Let the output of the security experiment be defined as 1 if $\mathcal{A}$'s output b' is equal to the challenge bit b; else let the output of the experiment be defined as 0. We will use a hybrid argument to show that the output of the experiment is 1 with probability at most $1/2 + \mathsf{negl}(\lambda)$.

– *Hybrid* $\mathcal{H}_0$. This hybrid consists of the output of the experiment when run with adversary $\mathcal{A}$.

– *Hybrid* $\mathcal{H}_1$. This hybrid is identical to the previous hybrid, except that the secret key s_σ is sampled uniformly at random from $[B_{\mathsf{nidls}}]$ in $\mathsf{MKHSS.KeyGen}$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$.

  Proof. The only difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that in $\mathcal{H}_0$, the secret key s_σ is sampled uniformly at random from $[B_{\mathsf{sk}}]$ while in $\mathcal{H}_1$ it is sampled from $[B_{\mathsf{nidls}}]$. The indistinguishability of $\mathcal{H}_0$ from $\mathcal{H}_1$ reduces directly to the small exponent assumption with length B_{sk} (Definition 9). In particular, note that the MKHSS security experiment can be run using the small exponent assumption's challenge since $\mathsf{MKHSS.Share}$ only requires the public key f_σ. □

– *Hybrid* $\mathcal{H}_2$. This hybrid is identical to the previous hybrid, except that rct_1 is computed as $\mathsf{rct}_1 = (g^{u_1},\ g^{u_1'} \cdot g^{r_1})$, where $u_1, u_1' \leftarrow\$ [B_{\mathsf{nidls}}]$ and r_1 is the randomness used to compute ct_1.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$.

  Proof. The only difference between $\mathcal{H}_1$ and $\mathcal{H}_2$ is that in $\mathcal{H}_1$, $\mathsf{rct}_1 = (g^{u_1},\ f_\sigma^{u_1} \cdot g^{r_1})$ while in $\mathcal{H}_2$, $\mathsf{rct}_1 = (g^{u_1},\ g^{u_1'} \cdot g^{r_1})$, where $u_1, u_1', s_\sigma \leftarrow\$ [B_{\mathsf{nidls}}]$, $f_\sigma = g^{-s_\sigma}$, and r_1 is the randomness used to compute ct_1. Indistinguishability between $\mathcal{H}_1$ and $\mathcal{H}_2$ can thus be reduced directly to the DDH assumption in the NIDLS group (cf. Definition 8). □

– *Hybrid* $\mathcal{H}_3$. This hybrid is identical to the previous hybrid, except that rct_1 is computed as $\mathsf{rct}_1 = (g^{u_1},\ g^{u_1'})$ where $u_1, u_1' \leftarrow\$ [B_{\mathsf{nidls}}]$.

  Claim. $\mathcal{H}_3 \overset{s}{\approx} \mathcal{H}_2$.

  Proof. The only difference between $\mathcal{H}_2$ and $\mathcal{H}_3$ is that, in $\mathcal{H}_2$, $\mathsf{rct}_1 = (g^{u_1},\ g^{u_1'} \cdot g^{r_1})$ while in $\mathcal{H}_3$, $\mathsf{rct}_1 = (g^{u_1},\ g^{u_1'})$ where $u_1, u_1' \leftarrow\$ [B_{\mathsf{nidls}}]$ and r_1 is the randomness used to compute ct_1. However, from the definition of the NIDLS framework (cf. Definition 5), we have that $g^{u_1'}$ is statistically close to the uniform distribution over the subgroup generated by g, which in turn implies that $g^{u_1'} \cdot g^{r_1}$ is also statistically close to the uniform distribution over the subgroup generated by g. It thus follows that $\mathcal{H}_2 \overset{s}{\approx} \mathcal{H}_3$. □

– *Hybrid* $\mathcal{H}_4$. This hybrid is identical to the previous hybrid, except that ct_1 is computed as $\mathsf{ct}_1 = (g^{r_1} \cdot h^{x_b},\ g^{r_1'})$, where $r_1, r_1' \leftarrow\$ [B_{\mathsf{nidls}}]$.

  Claim. $\mathcal{H}_4 \approx_c \mathcal{H}_3$.

  Proof. The only difference between $\mathcal{H}_3$ and $\mathcal{H}_4$ is that in $\mathcal{H}_3$, $\mathsf{ct}_1 = (g^{r_1} \cdot h^{x_b},\ f_\sigma^{r_1})$ while in $\mathcal{H}_4$, $\mathsf{ct}_1 = (g^{r_1} \cdot h^{x_b},\ g^{r_1'})$, where $r_1, r_1', s_\sigma \leftarrow\$ [B_{\mathsf{nidls}}]$ and $f_\sigma = g^{-s_\sigma}$. Indistinguishability between $\mathcal{H}_3$ and $\mathcal{H}_4$ can thus be reduced directly to the DDH assumption in the NIDLS group. □

– *Hybrid* $\mathcal{H}_5$. This hybrid is identical to the previous hybrid, except that ct_2 is computed as $\mathsf{ct}_2 = (g^{r_2},\ g^{r_2'} \cdot h^{x_b})$, where $r_2, r_2' \leftarrow\$ [B_{\mathsf{nidls}}]$.

Claim. $\mathcal{H}_5 \approx_c \mathcal{H}_4$.

Proof. The only difference between $\mathcal{H}_4$ and $\mathcal{H}_5$ is that in $\mathcal{H}_4$, $\mathsf{ct}_2 = (g^{r_2} \cdot h^{x_b},\ f_\sigma^{r_2})$ while in $\mathcal{H}_5$, $\mathsf{ct}_2 = (g^{r_2} \cdot h^{x_b},\ g^{r_2'})$, where $r_2, r_2', s_\sigma \leftarrow\$ [B_{\mathsf{nidls}}]$ and $f_\sigma = g^{-s_\sigma}$. Indistinguishability between $\mathcal{H}_4$ and $\mathcal{H}_5$ can thus be reduced directly to the DDH assumption in the NIDLS group. □

To complete the proof, observe that in $\mathcal{H}_5$ we have $[\![x_b]\!]^{\sigma}_{1-\sigma} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rct}_1)$, where $\mathsf{ct}_1 = (g^{r_1} \cdot h^{x_b},\ g^{r_1'})$, $\mathsf{ct}_2 = (g^{r_2},\ g^{r_2'} \cdot h^{x_b})$, and $\mathsf{rct}_1 = (g^{u_1},\ g^{u_1'})$, with $r_1, r_1', r_2, r_2', u_1, u_1' \leftarrow\$ [B_{\mathsf{nidls}}]$. Since $g^{r_1} \cdot h^{x_b}$ and $g^{r_2'} \cdot h^{x_b}$ are statistically close to the uniform distribution over the subgroup generated by g, the probability that the output of the experiment is 1 in $\mathcal{H}_5$, is at most $1/2 + \mathsf{negl}(\lambda)$. It follows that $\mathcal{H}_0 \approx_c \mathcal{H}_5$. Thus, $\mathcal{A}$ wins the MKHSS security game with probability of at most $1/2 + \mathsf{negl}(\lambda)$. ■

Instantiating the NIDLS group in Theorem 1 yields MKHSS schemes based on the DDH and small-exponent assumptions overs over the Paillier group, a class group, or an extension of the Joye–Libert group (cf. Section 3.1). In particular, the class group instantiation has transparent setup.

Remark 3 (Necessity of DDH over Paillier group). The security of the NIDLS-based HSS scheme in Abram et al. [ADOS22] can be reduced to the DCR assumption by instantiating the NIDLS framework with the ciphertext space of the Damgård–Jurik encryption scheme. Specifically, the Damgård–Jurik encryption scheme allows for a subgroup $\mathbb{H}$ of order t that is exponentially larger than B_{nidls}, ensuring that the small-exponent assumption holds unconditionally. Moreover, the security of the HSS construction reduces to the IND-CPA security of the NIDLS ElGamal encryption scheme, which in this case is secure under the DCR assumption (see Figure 18 and Lemma 5 for details). This raises a natural question: Is the MKHSS construction from Figure 5 secure assuming only DCR, when instantiated with the ciphertext space of the Damgård–Jurik encryption scheme?

However, the DDH assumption over the Damgård–Jurik group seems necessary in this case. Specifically, the security of the MKHSS construction relies on the security of the ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ and rct_1 computed in $\mathsf{MKHSS.Share}$. While the security of ct_1 and ct_2 can be reduced to the IND-CPA security of the NIDLS ElGamal encryption scheme, and thereby DCR, the security of rct_1 requires the DDH assumption. This is because it is an encryption of g^{r_1} instead of h^{r_1}. In fact, the IND-CPA security of ciphertexts of the form $(g^u, f^u \cdot g^x)$, where the plaintext x is encoded as the exponent of g as opposed to h, implies the security of DDH.

4.4 MKHSS from DDH

In this section, we construct multi-key HSS from DDH, closely following the construction described in the technical overview. We first define the DDH assumption and then recall the BHHO encryption scheme, which we extend slightly to simplify the presentation of our MKHSS construction.

Definition 10 (Decisional Diffie–Hellman Assumption). *Let* GGen *be a group generator such that* $\mathsf{GGen}(1^\lambda) \to (\mathbb{G}, p, g)$ *where* $\mathbb{G}$ *is a cyclic group of prime order* p *and* g *is a generator of* $\mathbb{G}$. *We also assume the existence of an efficient algorithm to compute over* $\mathbb{G}$. *The Decisional Diffie–Hellman (DDH) assumption is said to hold with respect to* GGen *if for all efficient adversaries, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, *we have*

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, p, g, g^x, g^y, g^{xy}) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, p, g, g^x, g^y, g^z) = 1\right]\right| \le \mathsf{negl}(\lambda),$$

where the probabilities are over the choice of $(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^\lambda)$ *and* $x, y, z \leftarrow\$ \mathbb{Z}_p$.

The following lemma, adapted from Boneh et al. [BHHO08], is an immediate consequence of the DDH assumption and will simplify the proofs of our constructions.

Lemma 4 (Matrix DDH [BHHO08]). *If the DDH assumption holds with respect to* GGen, *then for any polynomial* $\mathsf{poly}(\cdot)$ *and any efficient adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\cdot)$, *such that for all* $\lambda \in \mathbb{N}$, *all* $\ell_1, \ell_2 \in \mathbb{N}$, *where* $\ell_1, \ell_2 < \mathsf{poly}(\lambda)$, *we have*

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, p, g, \mathbf{g}_x, \mathbf{g}_y, \mathbf{g}_{xy}) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, p, g, \mathbf{g}_x, \mathbf{g}_y, \mathbf{g}_z) = 1\right]\right| \le \mathsf{negl}(\lambda),$$

where the probabilities are over the choice of $(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^\lambda)$*,* $\mathbf{x} = (x_1, \ldots, x_{\ell_1}) \leftarrow\$ \mathbb{Z}_p^{\ell_1}$*,* $\mathbf{y} = (y_1, \ldots, y_{\ell_2}) \leftarrow\$ \mathbb{Z}_p^{\ell_2}$*,* $\mathbf{z} = (z_1, \ldots, z_{\ell_1 \cdot \ell_2}) \leftarrow\$ \mathbb{Z}_p^{\ell_1 \cdot \ell_2}$*, and where* $\mathbf{g}_x = (g^{x_1}, \ldots, g^{x_{\ell_1}})$*,* $\mathbf{g}_y = (g^{y_1}, \ldots, g^{y_{\ell_2}})$*,* $\mathbf{g}_z = (g^{z_1}, \ldots, g^{z_{\ell_1 \cdot \ell_2}})$*, and* $\mathbf{g}_{xy} = (g^{x_1 \cdot y_1}, \ldots, g^{x_i \cdot y_j}, \ldots, g^{x_{\ell_1} \cdot y_{\ell_2}})$*.*

BHHO encryption. We recall the details of the BHHO encryption scheme [BHHO08] in Figure 7. Informally, BHHO is a circular secure variant of the ElGamal encryption scheme. Our description of the encryption scheme uses $\mathbb{G}$ as the message space, which simplifies the presentation of our MKHSS scheme by allowing us to use different elements in $\mathbb{G}$ as the base when encoding secrets in $\mathbb{Z}_p$. Similar to the NIDLS ElGamal scheme discussed in Section 4.3, we extend the BHHO scheme with a "flipped" encryption algorithm FlipEncrypt. This allows computing an encryption of x^{s_i} only using the public key, where x is the message and s_i is the i-th bit of the secret key. Specifically, for any ciphertext $\mathsf{ct} = (g_1^r, \ldots, g_{i-1}^r,\ g_i^r x,\ g_{i+1}^r, \ldots, g_{\ell_{\mathsf{sk}}}^r,\ f^r) \leftarrow \mathsf{FlipEncrypt}(\mathsf{crs}, \mathsf{pk}, i, x)$ we have

$$\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) = \langle \mathsf{ct}, \mathsf{sk} \rangle = x^{s_i} \cdot \prod_{j=1}^{\ell_{\mathsf{sk}}} g_j^{r s_j} \cdot f^r = x^{s_i}.$$

It can be shown that FlipEncrypt is CPA secure under the DDH assumption. However, to simplify the presentation, this is implicit in the proof of our MKHSS construction, where the security is directly reduced to DDH instead of the CPA security of the extended BHHO scheme.

BHHO Encryption Scheme

Public Parameters. A group generator GGen with respect to which the DDH assumption holds.

```
BHHO.Setup($1^\lambda$):
  1 : $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$
  2 : $\ell_{\mathsf{sk}} := \lceil 3 \log_2 p \rceil$
  3 : $g_1, \ldots, g_{\ell_{\mathsf{sk}}} \leftarrow\$ \mathbb{G}$
  4 : $\mathsf{crs} := (\mathbb{G}, g, \{g_i\}_{i=1}^{\ell_{\mathsf{sk}}}, p)$
  5 : return crs

BHHO.KeyGen(crs):
  1 : parse $\{g_i\}_{i=1}^{\ell_{\mathsf{sk}}}$ from crs
  2 : $s_1, \ldots, s_{\ell_{\mathsf{sk}}} \leftarrow\$ \{0,1\}$
  3 : $f := \prod_{i=1}^{\ell_{\mathsf{sk}}} g_i^{-s_i}$
  4 : $\mathsf{pk} := (g_1, \ldots, g_{\ell_{\mathsf{sk}}}, f)$
  5 : $\mathsf{sk} := (s_1, \ldots, s_{\ell_{\mathsf{sk}}}, 1)$
  6 : return (pk, sk)

BHHO.Encrypt(crs, pk, $x$):
  1 : parse $p$ from crs
  2 : parse $\mathsf{pk} = (g_1, \ldots, g_{\ell_{\mathsf{sk}}}, f)$
  3 : $r \leftarrow\$ \mathbb{Z}_p$
  4 : $\mathsf{ct} := (g_1^r, \ldots, g_{\ell_{\mathsf{sk}}}^r, f^r x)$
  5 : return ct

BHHO.FlipEncrypt(crs, pk, $i$, $x$):
  1 : parse $p$ from crs
  2 : parse $\mathsf{pk} = (g_1, \ldots, g_{\ell_{\mathsf{sk}}}, f)$
  3 : $r \leftarrow\$ \mathbb{Z}_p$
  4 : $\mathsf{ct} := (g_1^r, \ldots, g_{i-1}^r,\ g_i^r x,\ g_{i+1}^r, \ldots, g_{\ell_{\mathsf{sk}}}^r,\ f^r)$
  5 : return ct

BHHO.Decrypt(sk, ct):
  1 : $x := \langle \mathsf{ct}, \mathsf{sk} \rangle$
  2 : return $x$
```

Fig. 7: The BHHO encryption scheme.

DDH MKHSS construction. We present our MKHSS construction from DDH in Figure 8. The construction closely follows the description in the technical overview. Here, we also define two sub-procedures that capture synchronization of input shares (like we did in the NIDLS-based MKHSS construction).

Performance analysis. The share of each input sent to the other party comprises of $\Theta(\ell_{\mathsf{sk}}{}^3)$ group elements. Synchronizing each input share requires at most $O(\ell_{\mathsf{sk}}{}^3)$ group exponentiations, while evaluating each multiplication instruction requires $\Theta(\ell_{\mathsf{sk}}{}^2)$ group exponentiations and $\Theta(\ell_{\mathsf{sk}})$ distributed discrete logarithm computations. Compared to the DDH-based HSS scheme in [BGI16], the MKHSS scheme requires communicating $\Theta(\ell_{\mathsf{sk}})$ times more group elements per input. The evaluation cost of the MKHSS scheme is dominated by that of input share synchronization when the program contains $o(\ell_{\mathsf{sk}})$ multiplication instructions, beyond which it is only a constant factor slower than the HSS evaluation in [BGI16].

DDH-based MKHSS

Public Parameters. Let ε be an error bound, B_{size} be a bound on the RMS program size, let B be a magnitude bound, and let GGen be a group generator with respect to which the DDH assumption holds. We use the BHHO encryption scheme (cf. Figure 7) instantiated using GGen, DEval (cf. Figure 1) instantiated with an $\varepsilon/((\ell_{\mathsf{sk}}+1)\cdot B_{\mathsf{size}})$-correct, B-bounded, base-g DDLog algorithm, and the subroutines $\mathsf{ExpLinEncS}$ and $\mathsf{ExpLinEncR}$ defined in Figure 9.

Notation. Let $\mathbf{1}_{\mathbb{G}} = (1_{\mathbb{G}}, \ldots, 1_{\mathbb{G}}) \in \mathbb{G}^{\ell_{\mathsf{sk}}+1}$, where $1_{\mathbb{G}}$ is the identity, and let $\mathbf{0}_{\mathbb{Z}} = (0, \ldots, 0) \in \mathbb{Z}^{\ell_{\mathsf{sk}}+1}$.

```
$\mathsf{MKHSS.Setup}(1^\lambda)$:
  1 : $\mathsf{crs_{enc}} \leftarrow \mathsf{BHHO.Setup}(1^\lambda)$
  2 : $k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}} \leftarrow\$ \{0,1\}^\lambda$
  3 : $\mathsf{crs} := (\mathsf{crs_{enc}}, k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}})$
  4 : return $\mathsf{crs}$
```

```
$\mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$:
  1 : parse $\mathsf{crs_{enc}}$ from $\mathsf{crs}$
  2 : parse $g, \{g_i\}_{i=1}^{\ell_{\mathsf{sk}}}$, and $p$ from $\mathsf{crs_{enc}}$
  3 : parse $\mathsf{pk}_\sigma = (\mathsf{epk}_\sigma^{(0)}, \ldots, \mathsf{epk}_\sigma^{(\ell_{\mathsf{sk}})})$
  4 : for $i \in [\ell_{\mathsf{sk}}]$:
  5 :     $r_i, u_i \leftarrow\$ \mathbb{Z}_p$
  6 :     $\mathsf{ct}_i := \mathsf{FlipEncrypt}(\mathsf{crs_{enc}}, \mathsf{epk}_\sigma^{(0)}, i, g^x; r_i)$
  7 :     for $j \in [\ell_{\mathsf{sk}}]$:
  8 :         $\mathsf{rct}_{i,j} := \mathsf{Encrypt}(\mathsf{crs_{enc}}, \mathsf{epk}_\sigma^{(j)}, g_j^{r_i}; u_i)$
  9 : $\mathsf{ct}_{\ell_{\mathsf{sk}}+1} \leftarrow \mathsf{Encrypt}(\mathsf{crs_{enc}}, \mathsf{epk}_\sigma^{(0)}, g^x)$
 10 : $[\![x]\!]_\sigma^\sigma := (\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{(r_i, u_i)\}_{i=1}^{\ell_{\mathsf{sk}}})$
 11 : $[\![x]\!]_{1-\sigma}^\sigma := (\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{\mathsf{rct}_{i,j}\}_{1\le i,j\le \ell_{\mathsf{sk}}})$
 12 : return $([\![x]\!]_A^\sigma, [\![x]\!]_B^\sigma)$
```

```
$\mathsf{MKHSS.KeyGen}(\mathsf{crs})$:
  1 : parse $\mathsf{crs_{enc}}$ from $\mathsf{crs}$
  2 : parse $\{g_i\}_{i=1}^{\ell_{\mathsf{sk}}}, p$ from $\mathsf{crs_{enc}}$
  3 : $(\mathsf{epk}^{(0)}, \mathsf{esk}^{(0)}) \leftarrow \mathsf{BHHO.KeyGen}(\mathsf{crs_{enc}})$
  4 : $\boldsymbol{\gamma} := (\gamma^{(1)}, \ldots, \gamma^{(\ell_{\mathsf{sk}})}) \leftarrow\$ \mathbb{Z}_p^{\ell_{\mathsf{sk}}}$
  5 : $\Gamma := -\langle \boldsymbol{\gamma}, \mathsf{esk}^{(0)} \rangle$
  6 : for $i \in [\ell_{\mathsf{sk}}]$:
  7 :     $\mathsf{epk}^{(i)} := (g_i^{\gamma^{(1)}}, \ldots, g_i^{\gamma^{(\ell_{\mathsf{sk}})}}, g_i^{\Gamma})$
  8 : $\mathsf{pk} := (\mathsf{crs}, \mathsf{epk}^{(0)}, \ldots, \mathsf{epk}^{(\ell_{\mathsf{sk}})})$
  9 : $\mathsf{sk} := (\mathsf{pk}, \mathsf{esk}^{(0)}, \boldsymbol{\gamma})$
 10 : return $(\mathsf{pk}, \mathsf{sk})$
```

```
$\mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]_\sigma^A, [\![\mathbf{x}_B]\!]_\sigma^B, P)$:
  1 : parse $\mathsf{crs} = (\mathsf{crs_{enc}}, k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}})$
  2 : parse $\mathsf{esk}_\sigma^{(0)}$ from $\mathsf{sk}_\sigma$
  3 : parse $[\![\mathbf{x}_A]\!]_\sigma^A = ([\![x_A^{(1)}]\!]_\sigma^A, \ldots, [\![x_A^{(m)}]\!]_\sigma^A)$
  4 : parse $[\![\mathbf{x}_B]\!]_\sigma^B = ([\![x_B^{(1)}]\!]_\sigma^B, \ldots, [\![x_B^{(m)}]\!]_\sigma^B)$
  5 : for $i \in [m]$:
  6 :     $\{\!\{x_\sigma^{(i)}\}\!\} := \mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x_\sigma^{(i)}]\!]_\sigma^\sigma)$
  7 :     $\{\!\{x_{1-\sigma}^{(i)}\}\!\} := \mathsf{ExpLinEncR}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x_{1-\sigma}^{(i)}]\!]_\sigma^{1-\sigma})$
  8 : $\mathbf{k}_\sigma := \mathsf{esk}_\sigma^{(0)} \| \mathbf{0}_{\mathbb{Z}}$ if $\sigma = A$ else $\mathbf{0}_{\mathbb{Z}} \| (-\mathsf{esk}_\sigma^{(0)})$
  9 : $\mathsf{ek}_\sigma := (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \mathbf{k}_\sigma)$
 10 : $\{\!\{\mathbf{x}\}\!\} := (\{\!\{x_A^{(1)}\}\!\}, \ldots, \{\!\{x_A^{(m)}\}\!\}, \{\!\{x_B^{(1)}\}\!\}, \ldots, \{\!\{x_B^{(m)}\}\!\})$
 11 : return $\mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, \{\!\{\mathbf{x}\}\!\}, P)$
```

Fig. 8: MKHSS in arbitrary cyclic groups from DDH.

```
$\mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, \llbracket x \rrbracket^\sigma_\sigma)$:
  $\textbf{parse } \llbracket x \rrbracket^\sigma_\sigma \coloneqq (\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{(r_i, u_i)\}_{i=1}^{\ell_{\mathsf{sk}}})$
  $\textbf{parse } f_{1-\sigma} \textbf{ from } \mathsf{pk}_{1-\sigma}$
  $\textbf{parse } \boldsymbol{\gamma}_\sigma = (\gamma_\sigma^{(1)}, \ldots, \gamma_\sigma^{(\ell_{\mathsf{sk}})}), \mathsf{esk}_\sigma^{(0)} \textbf{ from } \mathsf{sk}_\sigma$
  $\varGamma_\sigma \coloneqq -\langle \boldsymbol{\gamma}_\sigma, \mathsf{esk}_\sigma^{(0)} \rangle$
  $\mathsf{pk}'_{1-\sigma} \coloneqq (f_{1-\sigma}^{\gamma_\sigma^{(1)}}, \ldots, f_{1-\sigma}^{\gamma_\sigma^{(\ell_{\mathsf{sk}})}}, f_{1-\sigma}^{\varGamma_\sigma})$
  $\textbf{for } i \in [\ell_{\mathsf{sk}}]:$
    $\mathsf{ct}'_i \coloneqq \mathsf{Encrypt}(\mathsf{crs}_{\mathsf{enc}}, \mathsf{pk}'_{1-\sigma}, f_{1-\sigma}^{r_i} \cdot f_\sigma^{-r_i}; u_i)$
    $\mathbf{c}_i^\sigma \coloneqq \mathsf{ct}_i \| \mathbf{1}_{\mathbb{G}} \textbf{ if } \sigma = A \textbf{ else } \mathbf{1}_{\mathbb{G}} \| \mathsf{ct}_i$
    $\mathbf{c}_i^{1-\sigma} \coloneqq \mathsf{ct}'_i \| \mathsf{ct}_i \textbf{ if } \sigma = A \textbf{ else } \mathsf{ct}_i \| \mathsf{ct}'_i$
  $\mathbf{c}^\sigma_{\ell_{\mathsf{sk}}+1} \coloneqq \mathsf{ct}_{\ell_{\mathsf{sk}}+1} \| \mathbf{1}_{\mathbb{G}} \textbf{ if } \sigma = A \textbf{ else } \mathbf{1}_{\mathbb{G}} \| \mathsf{ct}_{\ell_{\mathsf{sk}}+1}$
  $\mathbf{c}^{1-\sigma}_{\ell_{\mathsf{sk}}+1} \coloneqq \mathbf{c}^\sigma_{\ell_{\mathsf{sk}}+1}$
  $\{\!\{x\}\!\} \coloneqq (\mathbf{c}_1^A, \ldots, \mathbf{c}^A_{\ell_{\mathsf{sk}}+1}, \mathbf{c}_1^B, \ldots, \mathbf{c}^B_{\ell_{\mathsf{sk}}+1})$
  $\textbf{return } \{\!\{x\}\!\}$
```

```
$\mathsf{ExpLinEncR}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, \llbracket x \rrbracket^{1-\sigma}_\sigma)$:
  $\textbf{parse } \llbracket x \rrbracket^{1-\sigma}_\sigma \coloneqq (\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{\mathsf{rct}_{i,j}\}_{1 \le i,j \le \ell_{\mathsf{sk}}})$
  $\textbf{parse each } \mathsf{rct}_{i,j} = (\rho_{i,j}^{(1)}, \ldots, \rho_{i,j}^{(\ell_{\mathsf{sk}}+1)})$
  $\textbf{parse each } \mathsf{ct}_i = (\_, \ldots, \_, f_{1-\sigma}^{r_i})$
  $\textbf{parse } (s_\sigma^{(1)}, \ldots, s_\sigma^{(\ell_{\mathsf{sk}})}, 1) \textbf{ from } \mathsf{sk}_\sigma$
  $\textbf{for } i \in [\ell_{\mathsf{sk}}]:$
    $\textbf{for } k \in [\ell_{\mathsf{sk}} + 1]:$
      $\eta_i^{(k)} \coloneqq \prod_{j=1}^{\ell_{\mathsf{sk}}} (\rho_{i,j}^{(k)})^{-s_\sigma^{(j)}}$
    $\mathsf{ct}'_i \coloneqq (\eta_i^{(1)}, \ldots, \eta_i^{(\ell_{\mathsf{sk}}+1)} \cdot f_{1-\sigma}^{-r_i})$
    $\mathbf{c}_i^\sigma \coloneqq \mathsf{ct}_i \| \mathsf{ct}'_i \textbf{ if } \sigma = A \textbf{ else } \mathsf{ct}'_i \| \mathsf{ct}_i$
    $\mathbf{c}_i^{1-\sigma} \coloneqq \mathbf{1}_{\mathbb{G}} \| \mathsf{ct}_i \textbf{ if } \sigma = A \textbf{ else } \mathsf{ct}_i \| \mathbf{1}_{\mathbb{G}}$
  $\mathbf{c}^\sigma_{\ell_{\mathsf{sk}}+1} \coloneqq \mathbf{1}_{\mathbb{G}} \| \mathsf{ct}_{\ell_{\mathsf{sk}}+1} \textbf{ if } \sigma = A \textbf{ else } \mathsf{ct}_{\ell_{\mathsf{sk}}+1} \| \mathbf{1}_{\mathbb{G}}$
  $\mathbf{c}^{1-\sigma}_{\ell_{\mathsf{sk}}+1} \coloneqq \mathbf{c}^\sigma_{\ell_{\mathsf{sk}}+1}$
  $\{\!\{x\}\!\} \coloneqq (\mathbf{c}_1^A, \ldots, \mathbf{c}^A_{\ell_{\mathsf{sk}}+1}, \mathbf{c}_1^B, \ldots, \mathbf{c}^B_{\ell_{\mathsf{sk}}+1})$
  $\textbf{return } \{\!\{x\}\!\}$
```

Fig. 9: Exponent-linear encoding algorithms used as subroutines in the DDH MKHSS construction.

Theorem 2. *If the DDH assumption holds with respect to the group generator* GGen*, then for every polynomial* $\mathsf{poly}(\cdot)$ *and every error bound* $\varepsilon > 0$*, the construction described in Figure 8 is an* ε*-correct MKHSS scheme for the class of RMS programs with bound* B *and size at most* B_{size}*, and message space* $\mathbb{Z}_B$*, where* $1/\varepsilon, B_{\mathsf{size}}, B \le \mathsf{poly}(\lambda)$ *and* λ *is the security parameter of the MKHSS scheme.*

Proof. We first show that the construction satisfies the correctness property and then proceed to argue its security. We will use $\ell_{\mathsf{ct}} = \ell_{\mathsf{sk}} + 1$ as a shorthand to denote the length of ciphertexts.

Correctness. The correctness property requires that parties obtain a subtractive sharing of the program output upon evaluation. In our construction, parties run $\mathsf{ExpLinEncS}$ and $\mathsf{ExpLinEncR}$ to obtain an encoding of the input that is exponent-linear decodable (Definition 3) under the decoding key $\mathbf{k} = \mathsf{esk}_A^{(0)} \| \mathsf{esk}_B^{(0)}$. The parties then use the DEval algorithm with a trivial subtractive sharing of $\mathbf{k}$ and the DDLog algorithm from [BGI16] to evaluate the RMS program in a distributed manner. Correctness of the MKHSS construction then immediately follows from the correctness of DEval.

We now proceed to prove that the MKHSS construction has ε-correctness as defined in Definition 6. We first show that for an input x shared by party-σ, where $\sigma \in \{A, B\}$, the parties obtain the same output $\{\!\{x\}\!\}$, when party-σ runs $\mathsf{ExpLinEncS}$ on its share $\llbracket x \rrbracket^\sigma_\sigma$ and party-$(1-\sigma)$ runs $\mathsf{ExpLinEncR}$ on its share $\llbracket x \rrbracket^\sigma_{1-\sigma}$. Moreover, we prove that $\{\!\{x\}\!\}$ is exponent-linear decodable under the decoding key $\mathbf{k} = \mathsf{esk}_A^{(0)} \| \mathsf{esk}_B^{(0)}$.

Claim. For all integers $x \in \mathbb{Z}_B$ and all $\sigma \in \{A, B\}$, we have

$$\{\!\{x\}\!\} = \mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, \llbracket x \rrbracket^\sigma_\sigma) = \mathsf{ExpLinEncR}(\mathsf{sk}_{1-\sigma}, \mathsf{pk}_\sigma, \llbracket x \rrbracket^\sigma_{1-\sigma}),$$

where $(\llbracket x \rrbracket^\sigma_A, \llbracket x \rrbracket^\sigma_B) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$. Moreover, $\{\!\{x\}\!\}$ is base-g exponent-linear decodable under the decoding key $\mathbf{k} = \mathsf{esk}_A^{(0)} \| \mathsf{esk}_B^{(0)}$.

Proof. We consider the case when $\sigma = A$ for ease of exposition; a similar argument follows for the case when $\sigma = B$. From the description of $\mathsf{MKHSS.Share}$, we have $\llbracket x \rrbracket^A_A = \left(\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{(r_i, u_i)\}_{i=1}^{\ell_{\mathsf{sk}}} \right)$ and $\llbracket x \rrbracket^A_B = \left(\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{\mathsf{rct}_{i,j}\}_{1 \le i,j \le \ell_{\mathsf{sk}}} \right)$, where

$$\mathsf{ct}_i = (g_1^{r_i},\ \ldots,\ g_{i-1}^{r_i},\ g_i^{r_i} \cdot g^x,\ g_{i+1}^{r_i},\ \ldots,\ g_{\ell_{\mathsf{sk}}}^{r_i},\ f_A^{r_i})$$

$$\mathsf{rct}_{i,j} = (g_j^{\gamma_A^{(1)} \cdot u_i},\ \ldots,\ g_j^{\gamma_A^{(\ell_{\mathsf{sk}})} \cdot u_i},\ g_j^{\varGamma_A \cdot u_i} \cdot g_j^{r_i})$$

for each $i, j \in [\ell_{\mathsf{sk}}]$, and

$$\mathsf{ct}_{\ell_{\mathsf{sk}}+1} = (g_1^{r_{\ell_{\mathsf{ct}}}},\ \ldots,\ g_{\ell_{\mathsf{sk}}}^{r_{\ell_{\mathsf{ct}}}},\ f_A^{r_{\ell_{\mathsf{ct}}}} \cdot g^x).$$

We will first show that party-A and party-B obtain the same $\{\!\{x\}\!\} = (\mathbf{c}_1^A, \ldots, \mathbf{c}_{\ell_{\mathsf{sk}}+1}^A, \mathbf{c}_1^B, \ldots, \mathbf{c}_{\ell_{\mathsf{sk}}+1}^B)$. Observe that both parties compute $\mathbf{c}_i^A = \mathsf{ct}_i \| \mathbf{1}_{\mathbb{G}}$, for each $i \in [\ell_{\mathsf{sk}}+1]$, and compute $\mathbf{c}_{\ell_{\mathsf{sk}}+1}^B = \mathsf{ct}_{\ell_{\mathsf{sk}}+1} \| \mathbf{1}_{\mathbb{G}}$. However, in case of $\mathbf{c}_i^B$, where $i \in [\ell_{\mathsf{sk}}]$, each party locally computes a ct_i' and sets $\mathbf{c}_i^B = \mathsf{ct}_i' \| \mathsf{ct}_i$. We will argue that ct_i' computed by both parties are identical, which will in turn prove that they obtain the same $\{\!\{x\}\!\}$.

Consider any arbitrary $i \in [\ell_{\mathsf{sk}}]$. Party-$A$ computes ct_i' as

$$\begin{aligned}\mathsf{ct}_i' &= \mathsf{Encrypt}(\mathsf{crs}_{\mathsf{enc}}, \mathsf{pk}_B',\ f_B^{r_i} \cdot f_A^{-r_i}; u_i) \\ &= \left(f_B^{\gamma_A^{(1)} \cdot u_i},\ \ldots,\ f_B^{\gamma_A^{(\ell_{\mathsf{sk}})} \cdot u_i},\ f_B^{\mathit{\Gamma}_A \cdot u_i} \cdot f_B^{r_i} \cdot f_A^{-r_i} \right).\end{aligned}$$

On the other hand, party-B computes ct_i' as

$$\begin{aligned}\mathsf{ct}_i' &= \left(\prod_{j=1}^{\ell_{\mathsf{sk}}} \left(g_j^{\gamma_A^{(1)} \cdot u_i} \right)^{-s_B^{(j)}},\ \ldots,\ \prod_{j=1}^{\ell_{\mathsf{sk}}} \left(g_j^{\gamma_A^{(\ell_{\mathsf{sk}})} \cdot u_i} \right)^{-s_B^{(j)}},\ \prod_{j=1}^{\ell_{\mathsf{sk}}} \left(g_j^{\mathit{\Gamma}_A \cdot u_i} \cdot g_j^{r_i} \right)^{-s_B^{(j)}} \cdot f_A^{-r_i} \right) \\ &= \left(\left(\prod_{j=1}^{\ell_{\mathsf{sk}}} g_j^{-s_B^{(j)}} \right)^{\gamma_A^{(1)} \cdot u_i},\ \ldots,\ \left(\prod_{j=1}^{\ell_{\mathsf{sk}}} g_j^{-s_B^{(j)}} \right)^{\gamma_A^{(\ell_{\mathsf{sk}})} \cdot u_i},\ \left(\prod_{j=1}^{\ell_{\mathsf{sk}}} g_j^{-s_B^{(j)}} \right)^{\mathit{\Gamma}_A \cdot u_i + r_i} \cdot f_A^{-r_i} \right) \\ &= \left(f_B^{\gamma_A^{(1)} \cdot u_i},\ \ldots,\ f_B^{\gamma_A^{(\ell_{\mathsf{sk}})} \cdot u_i},\ f_B^{\mathit{\Gamma}_A \cdot u_i} \cdot f_B^{r_i} \cdot f_A^{-r_i}, \right)\end{aligned}$$

where the third equality follows from the fact that $f_B = \prod_{j=1}^{\ell_{\mathsf{sk}}} g_j^{-s_B^{(j)}}$. Thus, for each $i \in [\ell_{\mathsf{sk}}]$, both parties compute the same ct_i' and hence obtain identical outputs $\{\!\{x\}\!\}$.

We are left to show that $\{\!\{x\}\!\}$ is base-g exponent-linear decodable under

$$\mathbf{k} = (k_1, \ldots, k_{2\ell_{\mathsf{sk}}+2}) = \mathsf{esk}_A^{(0)} \| \mathsf{esk}_B^{(0)} = (s_A^{(1)}, \ldots, s_A^{(\ell_{\mathsf{sk}})}, 1, s_B^{(1)}, \ldots, s_B^{(\ell_{\mathsf{sk}})}, 1).$$

Observe that ct_i is an encryption of $g^{x \cdot s_A^{(i)}}$ under the public key $\mathsf{epk}_A^{(0)}$ for each $i \in [\ell_{\mathsf{sk}}]$, since they are computed using $\mathsf{FlipEncrypt}$. Similarly, $\mathsf{ct}_{\ell_{\mathsf{sk}}+1}$ is an encryption of g^x under the public key $\mathsf{epk}_A^{(0)}$ since it is computed using $\mathsf{Encrypt}$. This implies that $\left\langle \mathsf{ct}_i, \mathsf{esk}_A^{(0)} \right\rangle = g^{k_i}$ for each $i \in [\ell_{\mathsf{sk}}+1]$. However, since $\mathbf{c}_i^A = \mathsf{ct}_i \| \mathbf{1}_{\mathbb{G}}$, we have

$$\left\langle \mathbf{c}_i^A, \mathbf{k} \right\rangle = \left\langle \mathsf{ct}_i, \mathsf{esk}_A^{(0)} \right\rangle = g^{x \cdot k_i},$$

for each $i \in [\ell_{\mathsf{sk}}]$. Moreover, since $k_{2\ell_{\mathsf{sk}}+2} = k_{\ell_{\mathsf{sk}}+1} = 1$ and $\mathbf{c}_{\ell_{\mathsf{sk}}+1}^B = \mathbf{c}_{\ell_{\mathsf{sk}}+1}^A$, we have $\left\langle \mathbf{c}_{\ell_{\mathsf{sk}}+1}^B, \mathbf{k} \right\rangle = \left\langle \mathbf{c}_{\ell_{\mathsf{sk}}+1}^A, \mathbf{k} \right\rangle = g^x$.

Finally, for each $i \in [\ell_{\mathsf{sk}}]$ we have $\left\langle \mathbf{c}_i^B, \mathbf{k} \right\rangle = \left\langle \mathsf{ct}_i', \mathsf{esk}_A^{(0)} \right\rangle \cdot \left\langle \mathsf{ct}_i, \mathsf{esk}_B^{(0)} \right\rangle$. Here,

$$\begin{aligned}\left\langle \mathsf{ct}_i', \mathsf{esk}_A^{(0)} \right\rangle &= \left(\prod_{j=1}^{\ell_{\mathsf{sk}}} f_B^{\gamma_A^{(j)} \cdot u_i \cdot s_A^{(j)}} \right) \cdot f_B^{\mathit{\Gamma}_A \cdot u_i} \cdot f_B^{r_i} \cdot f_A^{-r_i} \\ &= f_B^{-\mathit{\Gamma}_A \cdot u_i} \cdot f_B^{\mathit{\Gamma}_A \cdot u_i} \cdot f_B^{r_i} \cdot f_A^{-r_i} \\ &= f_B^{r_i} \cdot f_A^{-r_i},\end{aligned}$$

where the second equality follows from the fact that $\mathit{\Gamma}_A = -\sum_{j=1}^{\ell_{\mathsf{sk}}} \gamma_A^{(j)} \cdot s_A^{(j)}$. Moreover, we have

$$\left\langle \mathsf{ct}_i, \mathsf{esk}_B^{(0)} \right\rangle = \left(\prod_{j=1}^{\ell_{\mathsf{sk}}} g_j^{r_i \cdot s_B^{(j)}} \right) \cdot g^{x \cdot s_B^{(i)}} \cdot f_A^{r_i} = g^{x \cdot s_B^{(i)}} \cdot f_B^{-r_i} \cdot f_A^{r_i},$$

where the second equality follows from the fact that $f_B = \prod_{j=1}^{\ell_{\mathsf{sk}}} g_j^{-s_B^{(j)}}$. It follows that

$$\left\langle \mathbf{c}_i^B, \mathbf{k} \right\rangle = \left\langle \mathsf{ct}'_i, \mathsf{esk}_A^{(0)} \right\rangle \cdot \left\langle \mathsf{ct}_i, \mathsf{esk}_B^{(0)} \right\rangle = g^{x \cdot s_B^{(i)}}.$$

Thus, $\{\!\{x\}\!\}$ is indeed base-g exponent-linear decodable under $\mathbf{k}$. □

In sum, it follows from the above claim that parties run DEval with encodings of the input that are base-g exponent-linear decodable. Additionally, observe that $\mathbf{k}_A$ and $\mathbf{k}_B$, computed by party-A and party-B in $\mathsf{MKHSS.Eval}$, form a subtractive sharing of $\mathbf{k}$ because $\mathbf{k}_A - \mathbf{k}_B = \left(\mathsf{esk}_A^{(0)} \,\|\, \mathbf{0}_{\mathbb{Z}}\right) - \left(\mathbf{0}_{\mathbb{Z}} \,\|\, -\mathsf{esk}_B^{(0)}\right) = \mathsf{esk}_A^{(0)} \| \mathsf{esk}_B^{(0)} = \mathbf{k}$. Let $\varepsilon' = \varepsilon / ((\ell_{\mathsf{sk}} + 1) \cdot B_{\mathsf{size}})$. Since B, B_{size} and $1/\varepsilon$ are all bounded by $\mathsf{poly}(\lambda)$ and ℓ_{sk} is polynomial in λ, we have ε' is polynomial in λ, which in turn implies from Lemma 2 that there exists an ε'-correct, B-bounded base-g algorithm for distributed discrete logarithm, as required by Figure 8. Furthermore, we have $s_A^{(i)}, s_B^{(i)} \le 1$ for each $i \in [\ell_{\mathsf{sk}}]$, and so it follows from Lemma 3 that the parties obtain a subtractive sharing of the program output upon MKHSS evaluation, except with a probability of at most $\varepsilon' \cdot B_{\mathsf{size}} \cdot (\ell_{\mathsf{sk}} + 1) + \mathsf{negl}(\lambda) = \varepsilon + \mathsf{negl}(\lambda)$. Thus, the MKHSS scheme has ε-correctness.

Security. The security property requires that the input share $[\![x]\!]_{1-\sigma}^{\sigma}$ of party-$(1-\sigma)$, ensures the privacy of an input x shared using party-σ's public key pk_σ. Recall that the public key pk_σ consists of $\mathsf{pk}_\sigma = (\mathsf{epk}_\sigma^{(0)}, \ldots, \mathsf{epk}_\sigma^{(\ell_{\mathsf{sk}})})$ where

$$\begin{aligned} \mathsf{epk}_\sigma^{(0)} &= (g_1,\ \ldots,\ g_{\ell_{\mathsf{sk}}},\ f_\sigma), \\ \mathsf{epk}_\sigma^{(i)} &= \left(g_i^{\gamma_\sigma^{(1)}},\ \ldots,\ g_i^{\gamma_\sigma^{(\ell_{\mathsf{sk}})}},\ g_i^{\Gamma_\sigma}\right), \quad \forall i \in [\ell_{\mathsf{sk}}]. \end{aligned}$$

while the share $[\![x]\!]_{1-\sigma}^{\sigma} = \left(\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{\mathsf{rct}_{i,j}\}_{1 \le i,j \le \ell_{\mathsf{sk}}}\right)$ where

$$\begin{aligned} \mathsf{ct}_i &= (g_1^{r_i},\ \ldots,\ g_{i-1}^{r_i},\ g_i^{r_i} \cdot g^x,\ g_{i+1}^{r_i},\ \ldots,\ g_{\ell_{\mathsf{sk}}}^{r_i},\ f_\sigma^{r_i}) \\ \mathsf{rct}_{i,j} &= (g_j^{\gamma_\sigma^{(1)} \cdot u_i},\ \ldots,\ g_j^{\gamma_\sigma^{(\ell_{\mathsf{sk}})} \cdot u_i},\ g_j^{\Gamma_\sigma \cdot u_i} \cdot g_j^{r_i}), \end{aligned}$$

for each $i, j \in [\ell_{\mathsf{sk}}]$, and

$$\mathsf{ct}_{\ell_{\mathsf{sk}}+1} = (g_1^{r_{\ell_{\mathsf{ct}}}},\ \ldots,\ g_{\ell_{\mathsf{sk}}}^{r_{\ell_{\mathsf{ct}}}},\ f_\sigma^{r_{\ell_{\mathsf{ct}}}} \cdot g^x).$$

As described in $\mathsf{MKHSS.Share}$, each ct_i is an encryption of $g^{x \cdot s_\sigma^{(i)}}$ using randomness r_i and each $\mathsf{rct}_{i,j}$ is an encryption of $g_j^{r_i}$. To argue security, we will first show that each $\mathsf{rct}_{i,j}$ is indistinguishable from a tuple of $\ell_{\mathsf{sk}} + 1$ random group elements, despite using the same randomness u_i to compute $\mathsf{rct}_{i,1}, \ldots, \mathsf{rct}_{i,\ell_{\mathsf{sk}}}$. This allows sampling $\mathsf{rct}_{i,j}$ independently of ct_i which, in turn, allows leveraging the security of the encryption scheme to argue the indistinguishability of ct_i. However, before we argue the indistinguishability of $\mathsf{rct}_{i,j}$, we will prove that the public keys $\mathsf{epk}_\sigma^{(0)}, \ldots, \mathsf{epk}_\sigma^{(\ell_{\mathsf{sk}})}$ can each be sampled independently.

We now proceed to prove the security of the MKHSS scheme. Consider any efficient adversary $\mathcal{A}$ for the security experiment defined in Definition 6. Let the output of the security experiment be defined as 1 if $\mathcal{A}$'s output b' is equal to the challenge bit b; else let the output of the experiment be defined as 0. We will use a hybrid argument to show that the output of the experiment is 1 with probability of at most $1/2 + \mathsf{negl}(\lambda)$.

– *Hybrid* $\mathcal{H}_0$. This hybrid is the output of the experiment when run with adversary $\mathcal{A}$.

– *Hybrid* $\mathcal{H}_1$. This hybrid is identical to the previous hybrid, except that $\mathsf{epk}_\sigma^{(i)}$ is computed as $\mathsf{epk}_\sigma^{(i)} = \left(g_1^{\eta_i}, \ldots, g_{\ell_{\mathsf{sk}}}^{\eta_i}, f_\sigma^{\eta_i}\right)$ for each $i \in [\ell_{\mathsf{sk}}]$ where $\eta_1, \ldots, \eta_{\ell_{\mathsf{sk}}}$ are uniformly random over $\mathbb{Z}_p$.

Claim. $\mathcal{H}_0 \overset{c}{\approx} \mathcal{H}_1$.

Proof. In $\mathcal{H}_0$, the public key pk_σ is of the form

$$\begin{aligned}\mathsf{epk}_\sigma^{(0)} &= (g_1, \ldots, g_{\ell_{\mathsf{sk}}}, f_\sigma) \\ \mathsf{epk}_\sigma^{(i)} &= \left(g_i^{\gamma_\sigma^{(1)}}, \ldots, g_i^{\gamma_\sigma^{(\ell_{\mathsf{sk}})}}, g_i^{\Gamma_\sigma}\right), \quad \forall i \in [\ell_{\mathsf{sk}}],\end{aligned}$$

while in this hybrid, pk_σ is of the form

$$\begin{aligned}\mathsf{epk}_\sigma^{(0)} &= (g_1, \ldots, g_{\ell_{\mathsf{sk}}}, f_\sigma) \\ \mathsf{epk}_\sigma^{(i)} &= \left(g_1^{\eta_i}, \ldots, g_{\ell_{\mathsf{sk}}}^{\eta_i}, f_\sigma^{\eta_i}\right), \quad \forall i \in [\ell_{\mathsf{sk}}],\end{aligned}$$

where $(\gamma_\sigma^{(1)}, \ldots, \gamma_\sigma^{(\ell_{\mathsf{sk}})})$ and $(\eta_1, \ldots, \eta_{\ell_{\mathsf{sk}}})$ are uniformly random over $\mathbb{Z}_p^{\ell_{\mathsf{sk}}}$. Note that the last component of each $\mathsf{epk}_\sigma^{(i)}$ is computed similarly in both hybrids—namely as the inner product of the first ℓ_{sk} elements of $\mathsf{epk}_\sigma^{(i)}$ with $\mathsf{esk}_\sigma^{(0)}$. The ciphertexts too are computed identically using the corresponding public keys in both hybrids. Thus, the only difference in the hybrids is in the distribution of the first ℓ_{sk} elements of each $\mathsf{epk}_\sigma^{(i)}$.
We will argue that $\mathcal{H}_0 \overset{c}{\approx} \mathcal{H}_1$ using a hybrid argument. Indistinguishability of $\mathcal{H}_0$ and $\mathcal{H}_1$ follows from Lemma 4, since the first ℓ_{sk} columns of $\mathsf{epk}_\sigma^{(i)}$ in $\mathcal{H}_0$ and $\mathcal{H}_1$ are each indistinguishable from a uniformly random matrix in $\mathbb{G}^{\ell_{\mathsf{sk}} \times (\ell_{\mathsf{sk}}+1)}$.

- *Hybrid $\mathcal{H}_{0.0}$.* This hybrid is identical to $\mathcal{H}_0$.
- *Hybrid $\mathcal{H}_{0.1}$.* This hybrid is identical to the previous hybrid, except that each $\mathsf{epk}_\sigma^{(i)}$ is computed as $\mathsf{epk}_\sigma^{(i)} = \left(g^{\eta_i^{(1)}}, \ldots, g^{\eta_i^{(\ell_{\mathsf{sk}})}}, g^{-\sum_{j=1}^{\ell_{\mathsf{sk}}} \eta_i^{(j)} \cdot s_\sigma^{(j)}}\right)$, where $(\eta_i^{(1)}, \ldots, \eta_i^{(\ell_{\mathsf{sk}})}) \leftarrow\$\ \mathbb{Z}_p^{\ell_{\mathsf{sk}}}$ are sampled uniformly random at random.
  The only difference between the two hybrids is that in $\mathcal{H}_{0.0}$, the first ℓ_{sk} elements of $\mathsf{epk}_\sigma^{(i)}$ are of the form $\left(g_i^{\gamma_\sigma^{(1)}}, \ldots, g_i^{\gamma_\sigma^{(\ell_{\mathsf{sk}})}}\right)$, where $\gamma_\sigma^{(j)}$ is uniformly random over $\mathbb{Z}_p$, for each $j \in [\ell_{\mathsf{sk}}]$, while in $\mathcal{H}_{0.1}$ they are sampled uniformly at random from $\mathbb{G}$. It thus follows from Lemma 4 (Matrix DDH) that $\mathcal{H}_{0.0} \overset{c}{\approx} \mathcal{H}_{0.1}$.
- *Hybrid $\mathcal{H}_{0.2}$.* This is identical to $\mathcal{H}_1$.
  The only difference between the two hybrids is that in $\mathcal{H}_{0.1}$, the first ℓ_{sk} elements of $\mathsf{epk}_\sigma^{(i)}$ are sampled uniformly at random from $\mathbb{G}$ while in $\mathcal{H}_{0.2}$ they are of the form $\left(g_1^{\eta_i}, \ldots, g_{\ell_{\mathsf{sk}}}^{\eta_i}\right)$, where η_i is uniformly random over $\mathbb{Z}_p$. It thus follows from Lemma 4 that $\mathcal{H}_{0.1} \overset{c}{\approx} \mathcal{H}_{0.2}$.

Consequently, we have $\mathcal{H}_{0.0} \overset{c}{\approx} \mathcal{H}_{0.2}$, which in turn implies that $\mathcal{H}_0 \overset{c}{\approx} \mathcal{H}_1$. □

– *Hybrid $\mathcal{H}_2$.* This hybrid is identical to the previous hybrid, except that $f_\sigma \leftarrow\$\ \mathbb{G}$ is sampled uniformly at random in $\mathsf{MKHSS.KeyGen}$.

Claim. $\mathcal{H}_1 \overset{s}{\approx} \mathcal{H}_2$.

Proof. The primary difference between the two hybrids is that in $\mathcal{H}_1$, we have

$$\mathsf{epk}_\sigma^{(0)} = \left(g^{\alpha_1}, \ldots, g^{\alpha_{\ell_{\mathsf{sk}}}}, g^{\sum_{j=1}^{\ell_{\mathsf{sk}}} \alpha_i \cdot s_\sigma^{(j)}}\right)$$

while in $\mathcal{H}_2$ we have

$$\mathsf{epk}_\sigma^{(0)} = \left(g^{\alpha_1}, \ldots, g^{\alpha_{\ell_{\mathsf{sk}}}}, f_\sigma\right),$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{\ell_{\mathsf{sk}}}) \leftarrow\$\ \mathbb{Z}_p^{\ell_{\mathsf{sk}}}$ and $f_\sigma \leftarrow\$\ \mathbb{G}$. Note that each $\mathsf{epk}_\sigma^{(i)}$ is computed identically from $\mathsf{epk}_\sigma^{(0)}$ in both hybrids. Similarly, given a public key pk, the ciphertexts too are computed in the same manner in both hybrids.
We will show that $\mathcal{H}_1 \overset{s}{\approx} \mathcal{H}_2$ from the leftover hash lemma (cf. Lemma 1). In more detail, let $H_{\boldsymbol{\alpha}}(\mathbf{s}) \coloneqq -\langle \boldsymbol{\alpha}, \mathbf{s} \rangle$ for any $\boldsymbol{\alpha} \in \mathbb{Z}_p^{\ell_{\mathsf{sk}}}$ and $\mathbf{s} \in \{0,1\}^{\ell_{\mathsf{sk}}}$. The family of hash functions $\mathcal{H} = \{H_{\boldsymbol{\alpha}}\}$ from

the set $\mathcal{X} = \{0,1\}^{\ell_{\mathsf{sk}}}$ to the set $\mathcal{Y} = \mathbb{Z}_p$ is 2-universal, which implies that $(\boldsymbol{\alpha}, -\langle \boldsymbol{\alpha}, \mathbf{s}\rangle)$ is $\frac{1}{p}$-uniform in $\mathbb{Z}_p^{\ell_{\mathsf{sk}}+1}$ from Lemma 1 since

$$\sqrt{\frac{p}{4 \cdot 2^{\ell_{\mathsf{sk}}}}} \leq \sqrt{\frac{p}{4 \cdot p^3}} = \frac{1}{2 \cdot p} < \frac{1}{p},$$

where the second equality follows from the fact that $\ell_{\mathsf{sk}} = \lceil 3 \log_2 p \rceil$. The only difference between the two hybrids is that, in $\mathcal{H}_1$ we have $f_\sigma = g^{H_{\boldsymbol{\alpha}}(\mathsf{esk}_\sigma^{(0)})}$ while in this hybrid, f_σ is uniformly random over $\mathbb{G}$. This implies that $\mathcal{H}_1 \overset{s}{\approx} \mathcal{H}_2$ since $\left(\boldsymbol{\alpha}, H_{\boldsymbol{\alpha}}(\mathsf{esk}_\sigma^{(0)})\right)$ is $1/p$-uniform over $\mathbb{Z}_p^{\ell_{\mathsf{sk}}+1}$, and $1/p$ is negligible. □

– *Hybrid* $\mathcal{H}_3$. This hybrid is identical to the previous hybrid, except that $\mathsf{epk}_\sigma^{(i)} \leftarrow\$ \mathbb{G}^{\ell_{\mathsf{sk}}+1}$ is sampled uniformly at random, for each $i \in [\ell_{\mathsf{sk}}]$ in $\mathsf{MKHSS.KeyGen}$.

  Claim. $\mathcal{H}_2 \overset{c}{\approx} \mathcal{H}_3$.

  Proof. In both hybrids, $\mathsf{epk}_\sigma^{(0)} = (g_1, \ldots, g_{\ell_{\mathsf{sk}}}, f_\sigma)$ is sampled uniformly at random from $\mathbb{G}^{\ell_{\mathsf{sk}}+1}$. However, in $\mathcal{H}_2$ we have

$$\mathsf{epk}_\sigma^{(i)} = \left(g_1^{\eta_i},\ \ldots,\ g_{\ell_{\mathsf{sk}}}^{\eta_i},\ f_\sigma^{\eta_i}\right),$$

  for each $i \in [\ell_{\mathsf{sk}}]$ where $\eta_1, \ldots, \eta_{\ell_{\mathsf{sk}}}$ are uniformly random over $\mathbb{Z}_p$. On the other hand, in $\mathcal{H}_3$, each $\mathsf{epk}_\sigma^{(i)}$ is sampled uniformly at random from $\mathbb{G}^{\ell_{\mathsf{sk}}+1}$. Note that any differences in the ciphertexts $\mathsf{rct}_{i,j}$ only stem from the differences in the public keys. The ciphertexts are otherwise computed identically using the corresponding public keys in both hybrids. Thus, it follows from Lemma 4 that the two hybrids are indistinguishable. □

– *Hybrid* $\mathcal{H}_4$. This hybrid is identical to the previous hybrid, except that $\mathsf{rct}_{i,j} \leftarrow\$ \mathbb{G}^{\ell_{\mathsf{sk}}+1}$ is sampled uniformly at random for each $i, j \in [\ell_{\mathsf{sk}}]$ in $\mathsf{MKHSS.Share}$.

  Claim. $\mathcal{H}_3 \overset{c}{\approx} \mathcal{H}_4$.

  Proof. The only difference between the two hybrids is that in $\mathcal{H}_3$, $(\mathsf{rct}_{i,1}, \ldots, \mathsf{rct}_{i,\ell_{\mathsf{sk}}})$ are encryptions computed using the same randomness u_i while in $\mathcal{H}_4$ they are sampled uniformly at random. In more detail, in $\mathcal{H}_3$,

$$\mathsf{rct}_{i,j} = \left(g^{\gamma_j^{(1)} \cdot u_i}, \ldots, g^{\gamma_j^{(\ell_{\mathsf{sk}})} \cdot u_i}, g^{\gamma_j^{(\ell_{\mathsf{sk}}+1)} \cdot u_i} \cdot g_j^{r_i}\right),$$

  for each $i, j \in [\ell_{\mathsf{sk}}]$ where $\left(\gamma_j^{(1)}, \ldots, \gamma_j^{(\ell_{\mathsf{sk}}+1)}\right)$ is uniformly random over $\mathbb{Z}_p^{\ell_{\mathsf{sk}}+1}$, for each $j \in [\ell_{\mathsf{sk}}]$ and $(u_1, \ldots, u_{\ell_{\mathsf{sk}}})$ is uniformly random over $\mathbb{Z}_p^{\ell_{\mathsf{sk}}}$. On the other hand, in case of $\mathcal{H}_4$, all $\mathsf{rct}_{i,j}$ are uniformly random over $\mathbb{G}^{\ell_{\mathsf{sk}}+1}$. It thus follows from Lemma 4 that the two hybrids are indistinguishable. □

– *Hybrid* $\mathcal{H}_5$. This hybrid is identical to the previous hybrid, except that $\mathsf{ct}_i \leftarrow\$ \mathbb{G}^{\ell_{\mathsf{sk}}+1}$ is sampled uniformly at random for each $i \in [\ell_{\mathsf{sk}} + 1]$ in $\mathsf{MKHSS.Share}$.

  Claim. $\mathcal{H}_4 \overset{c}{\approx} \mathcal{H}_5$.

  Proof. The only difference between the two hybrids is that in $\mathcal{H}_4$,

$$\mathsf{ct}_i = (g^{\alpha_1 \cdot r_i}, \ldots, g^{\alpha_{i-1} \cdot r_i}, g^{\alpha_i \cdot r_i} \cdot g^{x_b}, g^{\alpha_{i+1} \cdot r_i}, \ldots, g^{\alpha_{\ell_{\mathsf{sk}}+1} \cdot r_i})$$

  for each $i \in [\ell_{\mathsf{sk}} + 1]$ where $(\alpha_1, \ldots, \alpha_{\ell_{\mathsf{sk}}+1})$ and $(r_1, \ldots, r_{\ell_{\mathsf{sk}}+1})$ are uniformly random over $\mathbb{Z}_p^{\ell_{\mathsf{sk}}+1}$, while in $\mathcal{H}_5$, each ct_i is uniformly random over $\mathbb{G}^{\ell_{\mathsf{sk}}+1}$. Indistinguishability of the two hybrids follows directly from Lemma 4. □

To complete the proof, observe that in $\mathcal{H}_5$, $[\![x_b]\!]_{1-\sigma}^{\sigma} = \left(\{\mathsf{ct}_i\}_{i=1}^{\ell_{\mathsf{sk}}+1}, \{\mathsf{rct}_{i,j}\}_{1 \leq i,j \leq \ell_{\mathsf{sk}}}\right)$ where each ct_i and $\mathsf{rct}_{i,j}$ is sampled uniformly at random from $\mathbb{G}^{\ell_{\mathsf{sk}}+1}$. Thus, $[\![x_b]\!]_{1-\sigma}^{\sigma}$ is independent of x_b which implies that the output of the experiment in $\mathcal{H}_5$ is 1 with a probability of at most $1/2$. It follows from our argument above that $\mathcal{H}_0 \approx_c \mathcal{H}_5$, which implies that $\mathcal{A}$ wins the MKHSS security game with a probability of at most $1/2 + \mathsf{negl}(\lambda)$. ■

5 Applications

In this section, we describe direct applications of our MKHSS constructions.

5.1 Sublinear, two-round secure computation

In this section, we discuss how MKHSS can be applied to achieve sublinear, two-round, two-party secure computation protocols. Sublinear here means that the communication cost is bounded by a fixed polynomial in the total length of the inputs, outputs and the security parameter, but is independent of the circuit evaluated. We refer the reader to Goldreich [Gol06] for the standard security definitions of two-party secure computation.

Secure computation from DCR and NIDLS. An MKHSS scheme with negligible correctness error immediately implies a sublinear two-round secure computation protocol, as outlined in Section 1. Consequently, we obtain the following corollary of Theorems 1 and 6.

Corollary 1 (Sublinear protocols for RMS programs)**.** *There exists a sublinear two-round, two-party, secure computation protocol for evaluating polynomial-size RMS programs in the common reference string model and with semi-honest security, under either (1) the DCR assumption, or (2) the DDH and small exponent assumptions in the NIDLS framework.*

Secure computation from DDH. In contrast to MKHSS with a negligible correctness error, our DDH-based construction of MKHSS does not immediately imply a secure computation protocol, since it has noticeable correctness error. Fortunately, we can obtain a similar result using the same techniques as the ones used by Boyle, Gilboa, and Ishai [BGI17] to realize a sublinear, secure computation protocol from DDH-based HSS (which also has a noticeable correctness error).

In more detail, adopting the same approach as sketched in Section 1, which simply reconstructs the output in the second round of the protocol, leads to a security issue: learning that the output has an error leaks information about the inputs and the secret keys of the MKHSS scheme. The primary challenge in constructing the secure computation protocol is handling this leakage.

The template for leakage-resilience. We describe the template used by Boyle et al. [BGI17] to derive an analogous theorem for DDH-based HSS [BGI17, Theorem 4.14]. In particular, their template is general and does not exploit specific properties of the underlying DDH-based HSS scheme, making it also apply to our DDH-based MKHSS scheme.

As a first step, the idea is to use a simulatable, Las Vegas DDLog algorithm described in Boyle et al. [BGI17] to instantiate DEval, which limits the leakage to a bounded number γ of intermediate memory values within the computation of the RMS program and/or the bits of the secret key. Note that this does not affect the correctness or the security of the MKHSS scheme.

Next, to ensure security despite the leakage on intermediate values of the computation, the idea is to replace the MKHSS evaluation of the circuit C with an evaluation of a leakage-resilient circuit C' computing the same function. In a nutshell, C' emulates the execution of a multi-party secure computation protocol of C, which guarantees correctness and security against (up to) γ corruptions. This means that privacy of the inputs is preserved when up to γ intermediate values of C' (which correspond to the views of at most γ parties in the emulated multi-party protocol) are leaked.

Finally, to deal with the leakage on the bits of the secret key, each party needs to sample a sufficiently long BHHO secret key, such that leaking γ bits continues to ensure security of the MKHSS input shares. Intuitively, this is secure because of the leakage-resilience property of the BHHO encryption scheme [NS09].

We refer to Boyle et al. [BGI17] for full details on these different components and how they transform a leaky evaluation into a secure two-party protocol. Using the leakage-resilience template, we obtain the following corollary of Theorem 2.

Corollary 2 (Sublinear protocols for NC^1)**.** *Under the DDH assumption, there exists a sublinear two-round, two-party, secure protocol for evaluating NC^1 circuits in the common reference string model and with semi-honest security.*

Note that Corollary 2 only gives a secure computation protocol for NC^1 circuits C, as opposed to all RMS programs, because we require that the MKHSS scheme supports evaluation of the leakage resilient version of C. As discussed in Boyle et al. [BGI17], it is not known if RMS programs (or even branching programs) can be evaluated in a leakage-resilient manner using an RMS program.

5.2 Attribute-based non-interactive key exchange

An intriguing application of MKHSS is the ability to perform policy-based key-exchange. In particular, two parties, Alice and Bob, each have secret attributes x_A and x_B, respectively. For a public predicate C, Alice and Bob obtain the same secret key if and only if $C(x_A, x_B) = 0$ (the predicate is satisfied), and independently distributed keys otherwise. In this process, nothing about Bob's secret attribute is leaked to Alice, as from her perspective she always receives a random key, and vice versa.

Kolesnikov et al. [KKL$^+$16] present an *interactive* solution for this problem using using garbled circuits and supporting general predicates.[14] Many related notions of attribute-based key-exchange also exist, including witness-authenticated key exchange (see the overview of Melissaris [Mel22]). We also note that attribute-based key exchange generalizes the widely-used notion of password-authenticated key exchange, where the predicate essentially checks that Alice and Bob hold the same secret attribute (or password).

To the best of our knowledge, we construct the first attribute-based *non-interactive* key-exchange (ANIKE) protocol for NC^1 predicates in the standard model. In particular, we show that MKHSS for polynomial-size RMS programs implies ANIKE with predicates from the same function class.

We present a universally composable (UC) corruptible ideal functionality for attribute-based non-interactive key exchange, which we then instantiate using MKHSS. This is a more desirable security guarantee for key exchange than a weaker property-based definition, since it composes with other primitives (key exchange is often used as a building block in larger protocols).

The ideal functionality is defined as follows. In the initialization phase (which happens once), the functionality receives an attribute x from every party. The adversary is assumed to statically corrupt an arbitrary subset of these parties. In the key exchange phase (which is repeatable), and instantiated between a pair of parties Alice and Bob, the functionality receives a request from Alice and Bob, which consists of a predicate, and outputs a fresh key to each party. If repeated with the same predicate, the functionality outputs the same keys deterministically. The pair of keys output by the functionality are defined as follows, depending on whether the parties are honest or corrupted and whether the predicate is satisfied.

First we consider the case where both parties are honest:

– If Alice and Bob's attributes satisfy the predicate, the functionality samples a fresh key k which it sends to both parties.

– If their attributes do not satisfy the predicate, the functionality independently samples two keys k_A and k_B, then sends k_A to Alice and k_B to Bob.

Second, we consider the case where one of the parties is corrupted:

– If both parties' attributes satisfy the predicate, the adversary gets to specify the key k, which the functionality sends to both parties.

– If their attributes do not satisfy the predicate, the functionality samples a uniformly random key to send to the uncorrupted party.

We refer to Functionality 1 for the full specification and define the algorithms, properties, and key exchange protocol that we will instantiate in Definition 11.

Definition 11 (Attribute-Based Non-interactive Key Exchange)**.** *An attribute-based non-interactive key exchange (ANIKE) protocol with attribute space $\mathcal{X}$ consists of three efficient algorithms* (Setup, $\mathsf{AttrKeyGen}$, $\mathsf{AttrKeyDer}$)*, which are used to instantiate the two-party protocol described in Figure 10, and which have the following syntax:*

– $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs.

– $\mathsf{AttrKeyGen}(\mathsf{crs}, x) \to (\mathsf{st}_\sigma)$. *The randomized attribute encoding algorithm takes as input the CRS* crs *and an attribute $x \in \mathcal{X}$. It outputs a public encoding* pe *and private state* st.

– $\mathsf{AttrKeyDer}(\sigma, \mathsf{st}_\sigma, \mathsf{pe}_{1-\sigma}, C) \to k$. *The deterministic key derivation algorithm takes as input the party identifier $\sigma \in \{A, B\}$, the CRS* crs*, the party's secret state* st_σ*, the other party's public encoding* $\mathsf{pe}_{1-\sigma}$*, and a circuit C describing the predicate. It outputs a key k.*

Security. We say that the ANIKE protocol is secure if it realizes the corruptible ideal functionality described in Functionality 1 against a semi-honest adversary assuming an authenticated channel.

Attribute-Based Non-Interactive Key Exchange Protocol

$\boxed{\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)}$

Alice(crs, x_A)		**Bob**(crs, x_B)
$(\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{AttrKeyGen}(\mathsf{crs}, x_A)$		$(\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{AttrKeyGen}(\mathsf{crs}, x_B)$
	$\xrightarrow{\mathsf{pe}_A}$ $\xleftarrow{\mathsf{pe}_B}$	
$k \leftarrow \mathsf{AttrKeyDer}(A, \mathsf{st}_A, \mathsf{pe}_B, C)$		$k \leftarrow \mathsf{AttrKeyDer}(B, \mathsf{st}_B, \mathsf{pe}_A, C)$

Fig. 10: The ANIKE protocol using the algorithms specified in Definition 11.

Construction. Our construction is parameterized by an MKHSS scheme supporting polynomial-size RMS programs. We assume, without loss of generality, that the MKHSS message space $\mathcal{M}$ is a finite field $\mathbb{F}$, such that $|\mathbb{F}| \geq 2^\lambda$. Such a setup can always be achieved by working in the field extension of $\mathbb{F}_2$.

Remark 4. For simplicity, in our construction, we let the algorithm $\mathsf{AttrKeyGen}$ be parameterized by a party identifier $\sigma \in \{A, B\}$. This allows us to construct $\mathsf{AttrKeyGen}$ "asymmetrically" by having it be defined differently depending on whether Alice or Bob runs it. This change is without loss of generality, since the party-agnostic version of $\mathsf{AttrKeyGen}$ can be recovered by having the parties play both roles simultaneously and agree on their respective roles in the key-derivation phase.

To encode her attribute x_A, Alice maps it into the field $\mathbb{F}$. She also samples a PRF key K which will be used to generate pseudorandom shifts $\Delta_A \in \mathbb{F}$ in the key derivation phase. Specifically, these shifts are used to ensure the derived keys are uniformly random when the predicate C is unsatisfied.

Alice then samples MKHSS keys $(\mathsf{pk}_A, \mathsf{sk}_A)$ and computes shares $([\![K_A \| x_A]\!]^A_A, [\![K_A \| x_A]\!]^A_B)$ of $K_A \| x_A$. Her public encoding consists of her MKHSS public key pk_A and Bob's share $[\![K_A \| x_A]\!]^A_B$, while her private encoding consists of her MKHSS secret key sk_A and her own share $[\![K_A \| x_A]\!]^A_A$. Bob encodes his attribute x_B analogously.

Given her own private encoding and Bob's public encoding, Alice now holds her MKHSS secret key sk_A, Bob's MKHSS public key pk_B, and her shares $[\![K_A \| x_A]\!]^A_A$ and $[\![K_B \| x_B]\!]^B_A$. She homomorphically evaluates the program P_C, where P_C is defined as:

$$P_C(K_A || x_A, K_B || x_B) = \underbrace{F_{K_A}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C)}_{\Delta_A} \cdot \underbrace{F_{K_B}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C)}_{\Delta_B} \cdot C(x_A, x_B).$$

That is, P_C computes the predicate $C(x_A, x_B)$ and then multiplies the result by $\Delta_A \cdot \Delta_B$, derived from the PRF. Bob symmetrically evaluates his shares for the same program P_C using his MKHSS secret key sk_B and Alice's MKHSS public key pk_A. Thus, if $C(x_A, x_B) = 0$, Alice and Bob end up with subtractive shares of 0, i.e., the same key. On the other hand, if $C(x_A, x_B) \neq 0$, Alice and Bob end up with shares of $\Delta_A \cdot \Delta_B$, i.e., independent pseudorandom keys.

We refer to Figure 11 for a formal description of our construction.

Theorem 3. *Assuming the existence of an MKHSS scheme* MKHSS *for polynomial-size RMS programs and the existence of PRFs in* NC^1*, the construction described in Figure 11 is an attribute-based non-interactive key exchange supporting predicates described by polynomial-size RMS programs.*

Proof. We show that Figure 11 securely realizes the functionality $\mathcal{F}_{\mathsf{anike}}$ described in Functionality 1 by constructing a simulator which simulates the view of the corrupted party and interacts with $\mathcal{F}_{\mathsf{anike}}$ on behalf of the ideal adversary.

[14] They define attribute-based key exchange in a client-server model, which is equivalent to our notion.

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

Parties. The functionality is parameterized by a set of parties and an adversary $\mathcal{A}$ that statically corrupts an arbitrary subset of the parties.

Procedure. The functionality aborts if it receives any incorrectly formatted messages.

– *One-time initialization phase.*

```
1: Receive a message $(\texttt{Init}, \mathsf{id}_\sigma, x)$ containing an attribute $x$ from every party with identifier $\sigma$.
2: Initialize a lookup table of generated keys $T$.
3: Send ready to $\mathcal{A}$.
```

– *Repeatable key exchange phase between Alice and Bob.*

```
1: Receive a message $(\texttt{KeyAgree}, \mathsf{id}_B, C)$ from Alice and a message $(\texttt{KeyAgree}, \mathsf{id}_A, C)$ from Bob, where $C$ is a circuit describing a predicate.
2: Receive a message from $\mathcal{A}$, which is either empty, contains a key and an identifier $\sigma \in \{\mathsf{id}_A, \mathsf{id}_B\}$, or contains two keys and both identifiers.
3: If $(\mathsf{id}_A, \mathsf{id}_B, C) \in T$:
    3.1 Set $(k_A, k_B) := T[(\mathsf{id}_A, \mathsf{id}_B, C)]$.
4: Else if $\mathcal{A}$ sent an empty message, i.e., the Alice and Bob are both honest:
    4.1 If $C(x_A, x_B) = 0$: sample $k_A$ uniformly and set $k_B = k_A$.
    4.2 Else if $C(x_A, x_B) = 1$: sample $k_A, k_B$ uniformly and independently.
5: Else if $\mathcal{A}$ sent $k_\sigma$, i.e., party-$\sigma$ is corrupted and party $\overline{\sigma} \in \{A, B\} \setminus \{\sigma\}$ is honest:
    5.1 If $C(x_A, x_B) = 0$: set $k_{\overline{\sigma}} := k_\sigma$.
    5.2 Else if $C(x_A, x_B) = 1$: sample $k_{\overline{\sigma}}$ uniformly.
6: If $\mathcal{A}$ sent $k_A$ and $k_B$, i.e., both parties are corrupted:
    6.1 Do nothing.
7: Set $T[(\mathsf{id}_A, \mathsf{id}_B, C)] = (k_A, k_B)$.
8: Output $k_A$ to Alice and $k_B$ to Bob.
```

Functionality 1: Corruptible ideal functionality for attribute-based non-interactive key exchange.

– *Initialization phase.* For every corrupted party, the simulator obtains their attribute x and sends it to $\mathcal{F}_{\mathsf{anike}}$ on behalf of the ideal adversary.

  We now emulate a key derivation phase between two parties, Alice and Bob.

– *Case 1: Both parties are honest.* By correctness of MKHSS, we have that

$$k_A - k_B = P_C(\Delta_A \| x_A, \Delta_B \| x_B) = C(x_A, x_B) \cdot \Delta_A \cdot \Delta_B \in \mathbb{F}.$$

  Thus, if $C(x_A, x_B) = 0$, we have that $k_A = k_B$, with all but negligible probability. Moreover, the tuple (k_A, k_B) is computationally indistinguishable from a uniformly random tuple $(k_A, k_B) \in \mathbb{F} \times \mathbb{F}$ subject to $k_A = k_B$, by the external security Definition 7. As such, when the predicate is satisfied, k_A and k_B matches the output of the ideal functionality.

  On the other hand, if $C(x_A, x_B) \neq 1$, we have $k_A = k_B + C(x_A, x_B) \cdot \Delta_A \cdot \Delta_B$, where Δ_A and Δ_B are the secret pseudorandom shifts output by the PRF evaluated under independent keys (by the correctness of MKHSS and the definition of the evaluated program P_C).

  We proceed to show that (k_A, k_B) is computationally indistinguishable from a random tuple via a simple hybrid argument:

  • *Hybrid $\mathcal{H}_0$.* This hybrid consists of the tuple (k_A, k_B), as computed using AttrKeyDer by each party in Figure 11.

Attribute-Based Non-Interactive Key Exchange from MKHSS

Public Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be an MKHSS scheme with external security (cf. Definition 7) for polynomial-size RMS programs defined over the finite field $\mathbb{F}$, where $|\mathbb{F}| \geq 2^\lambda$. Let $F : \{0,1\}^\lambda \times \{0,1\}^\star \to \mathbb{F}$ be a PRF. Let $F_K : \{0,1\}^\star \to \mathbb{F}$ be a PRF with keys sampled from $\{0,1\}^\lambda$, such that $F_k(x)$ is computable by a polynomial-size RMS program over $\mathbb{F}$.

The evaluated program. Define P_C to be the program that, on input $K_A \| x_A, K_B \| x_B$ outputs $F_{K_A}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C) \cdot F_{K_B}(A_{\mathsf{id}} \| B_{\mathsf{id}} \| C) \cdot C(x_A, x_B)$, where C is the attribute predicate.

```
$\mathsf{NIKE.Setup}(1^\lambda)$:
  1 : $\mathsf{crs} \leftarrow \mathsf{MKHSS.Setup}(\lambda)$
  2 : return $\mathsf{crs}$
```

```
$\mathsf{NIKE.AttrKeyGen}(\mathsf{crs}, \sigma, x)$:
  1 : $K \leftarrow\$ \{0,1\}^\lambda$
  2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$
  3 : $([\![K\|x]\!]^\sigma_A, [\![K\|x]\!]^\sigma_B) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, (K\|x))$
  4 : $\mathsf{pe} := (\mathsf{pk}, [\![K\|x]\!]^\sigma_{1-\sigma})$
  5 : $\mathsf{st} := (\mathsf{sk}, [\![K\|x]\!]^\sigma_\sigma)$
  6 : return $(\mathsf{pe}, \mathsf{st})$
```

```
$\mathsf{NIKE.AttrKeyDer}(\sigma, \mathsf{st}_\sigma, \mathsf{pe}_{1-\sigma}, C)$:
  1 : parse $\mathsf{pe}_{1-\sigma} = (\mathsf{pk}_{1-\sigma}, [\![K_{1-\sigma}\|x_{1-\sigma}]\!]^{1-\sigma}_\sigma)$
  2 : parse $\mathsf{st}_\sigma = (\mathsf{sk}_\sigma, [\![K_\sigma\|x_\sigma]\!]^\sigma_\sigma)$
  3 : $k \leftarrow \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![K_A\|x_A]\!]^A_\sigma, [\![K_B\|x_B]\!]^B_\sigma, P_C)$
  4 : return $k$
```

Fig. 11: Attribute-based non-interactive key exchange from MKHSS.

- *Hybrid $\mathcal{H}_1$.* In this hybrid, we replace k_B with a uniformly random key and set $k_A = k_B + C(x_A, x_B) \cdot \Delta_A \cdot \Delta_B$. This hybrid is computationally indistinguishable from the previous one by the external security of MKHSS.

- *Hybrid $\mathcal{H}_2$.* In this hybrid, we sample (k_A, k_B) as a uniformly random tuple over $\mathbb{F} \times \mathbb{F}$. This hybrid is computationally indistinguishable from the previous one by the pseudorandomness of $\Delta_A \cdot \Delta_B$ (which are generated internally using the PRF). To see this, it suffices to note that, in $\mathcal{H}_1$, we already have that k_B is computationally indistinguishable from a random field element conditioned on k_A, since we can view $\Delta_A \cdot \Delta_B$ as being a uniformly random element.

At this point, it suffices to note that $\mathcal{H}_2$ is distributed identically to the output of the ideal functionality when the predicate is not satisfied. This concludes the proof for the case where both parties are honest.

– *Case 2: Alice is corrupted.* The simulator computes $(\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$ and $([\![\mathbf{0}]\!]^A_B, [\![\mathbf{0}]\!]^B_B) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, B, \mathsf{pk}_B, \mathbf{0})$, to define $\mathsf{pe}_B := (\mathsf{pk}_B, [\![\mathbf{0}]\!]^B_A)$, where $\mathbf{0} := 0^{\lambda+|x|}$. The simulated view of Alice consists of crs and pe_B. Eventually, the simulator recovers x_A and $\mathsf{st}_A = (\mathsf{sk}_A, [\![K_A\|x_A]\!]^A_A)$. It computes $k_A := \mathsf{MKHSS.Eval}(\mathsf{crs}, A, \mathsf{sk}_A, \mathsf{pk}_B, [\![K_A\|x_A]\!]^A_A, [\![\mathbf{0}]\!]^B_A)$ and sends k_A to the functionality on behalf of the ideal adversary. The functionality outputs k_A to Alice and k_B to Bob. By the security of MKHSS and a straightforward hybrid argument, the joint distribution of Bob's message and the output of both parties in the real world is indistinguishable from the simulation of Bob's message and the output of the ideal functionality.

– *Case 3: Bob is corrupted.* This case follows by symmetry.

■

Corollary 3. *Assuming the existence of PRFs in* NC^1*, there exists an attribute-based non-interactive key exchange supporting predicates described by polynomial-size RMS programs under either (1) the DCR assumption or (2) the DDH and small exponent assumptions in the NIDLS framework.*

6 Public-Key PCFs from MKHSS and Applications

Practical secure computation protocols are realized in the preprocessing model [DPSZ12]: During an "offline" preprocessing phase, the computing parties generate a large amount of *pseudorandom correlations* that are independent of any function. Then, during an online phase, the parties use the stored correlations to compute a function over their inputs in a secure protocol. The advantage of this model is that it pushes the bulk of the communication and computation costs of the function-dependent online phase to the preprocessing phase. Pseudorandom correlation generators (PCGs) and functions (PCFs) push this model of secure computation to the limit by allowing parties to locally expand a short key into a virtually unbounded amount of correlated randomness. However, traditional approaches still require the parties to run an interactive protocol to generate their key.

Public-key PCFs. A *public-key* PCF (PK-PCF) is a PCF equipped with a non-interactive key distribution protocol. Public-key PCFs were originally introduced in the work of Orlandi et al. [OSY21] and formalized in the recent work of Bui et al. [BCM$^+$24]. PK-PCFs are motivated by their direct application to secure computation in the preprocessing model.

Let $\mathcal{Y}$ be a correlation such that a secure access to random samples from $\mathcal{Y}$ enables efficient information-theoretic two-party computation (typically, $\mathcal{Y}$ can sample an oblivious transfer correlation, Beaver triples, authenticated Beaver triples, or other types of correlated randomness, depending on the application at hand). A PK-PCF for $\mathcal{Y}$ induces the following appealing template for communication-efficient secure two-party computation:

Non-interactive preprocessing. Ahead of time, all participants P_i of a secure computation network upload their PK-PCF public key pk_i to some public bulletin board.
Fast, online secure computation. Whenever two parties P_i and P_j want to securely compute a function, they can retrieve each other's public keys, non-interactively derive correlated PCF keys, and generate as many pseudorandom samples from $\mathcal{Y}$ as they need to enable a fast online phase for a two-party protocol in the correlated randomness model (e.g., GMW [GMW87] or SPDZ [DPSZ12]).

In this section, we construct PK-PCFs for all correlations computable by RMS programs over a finite ring. As another application of our construction, in Section 6.3, we show a protocol for generating *multi-party* correlations with quadratic improvement in communication relative to the prior constructions of PCFs.

6.1 Background: Public-key pseudorandom correlation functions

In this section, we provide some background and a formal definition of PK-PCFs, following the formalization of Bui et al. [BCM$^+$24].

The standard PCF (and PK-PCF) definition requires the correlation to be "reverse sampleable" which, roughly speaking, means that given any (possibly adversarially generated) share of the target correlation, the other (honest) share can be efficiently sampled. We note that all additive correlations, which will be the target of our constructions, are reverse sampleable.

Remark 5 (Notation). In this section, we will identify the two parties using indices 0 and 1, instead of letters A and B as we did in previous sections. This simplifies notation when we define *multi-party* PCFs in Section 6.3.

Definition 12 (Reverse-Sampleable Correlation)**.** *Let* λ *be a security parameter and* $n = n(\lambda) \in \mathsf{poly}(\lambda)$ *be an output length. Define two efficient algorithms* $\mathcal{Y}$ *and* $\mathsf{RSample}$ *with the following syntax:*

– $\mathcal{Y}(1^\lambda) \to (y^0, y^1)$*. The randomized correlation sampling algorithm takes as input the security parameter and outputs a pair* $(y^0, y^1) \in \{0,1\}^n \times \{0,1\}^n$ *defining a correlation.*
– $\mathsf{RSample}(1^\lambda, \sigma, y^\sigma) \to y^{1-\sigma}$*. The deterministic reverse-sampling algorithm takes as input the security parameter, an index* $\sigma \in \{0,1\}$*, and a string* $y^\sigma \in \{0,1\}^n$*. It outputs a string* $y^{1-\sigma} \in \{0,1\}^n$*.*

We say that $\mathcal{Y}$ defines a reverse-sampleable *correlation if for all $\sigma \in \{0,1\}$, it holds that:*

$$\left\{ (y^0, y^1) \;\middle|\; (y^0, y^1) \leftarrow \mathcal{Y}(1^\lambda) \right\} \approx_s \left\{ (y^0, y^1) \;\middle|\; \begin{array}{r} (\hat{y}^0, \hat{y}^1) \leftarrow \mathcal{Y}(1^\lambda) \\ y^\sigma := \hat{y}^\sigma \\ y^{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y^\sigma) \end{array} \right\}.$$

```
$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma),\ \forall \sigma \in \{0,1\}$
foreach $i \in [N]$:
    $x_i \leftarrow\$ \{0,1\}^n$
    $(y_i^0, y_i^1) \leftarrow \mathcal{Y}(1^\lambda)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, (x_i, y_i^0, y_i^1)_{i\in[N]})$
return $b$
```

```
$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma),\ \forall \sigma \in \{0,1\}$
$\mathsf{k}_\sigma := \mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_{1-\sigma}, \mathsf{sk}_\sigma),\ \forall \sigma \in \{0,1\}$
foreach $i \in [N]$:
    $x_i \leftarrow\$ \{0,1\}^n$
    $y_i^\sigma := \mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, \mathsf{k}_\sigma, x_i),\ \forall i \in \{0,1\}$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, (x_i, y_i^0, y_i^1)_{\sigma\in[N]})$
return $b$
```

Fig. 12: Pseudorandom $\mathcal{Y}$-correlated outputs for a weak PK-PCF.

```
$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_{\hat\sigma}, \mathsf{sk}_{\hat\sigma}) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \hat\sigma),\ \forall \hat\sigma \in \{0,1\}$
$\mathsf{k}_{1-\sigma} := \mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathsf{sk}_{1-\sigma})$
foreach $i \in [N]$:
    $x_i \leftarrow\$ \{0,1\}^n$
    $y_i^{1-\sigma} := \mathsf{pkPCF.Eval}(\mathsf{crs}, 1-\sigma, \mathsf{k}_{1-\sigma}, x_i)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, (x_i, y_i^{1-\sigma})_{i\in[N]})$
return $b$
```

```
$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{pkPCF.Setup}(1^\lambda)$
$(\mathsf{pk}_{\hat\sigma}, \mathsf{sk}_{\hat\sigma}) \leftarrow \mathsf{pkPCF.KeyGen}(\mathsf{crs}, \hat\sigma),\ \forall \hat\sigma \in \{0,1\}$
$\mathsf{k}_\sigma := \mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, \mathsf{sk}_{1-\sigma})$
foreach $i \in [N]$:
    $x_i \leftarrow\$ \{0,1\}^n$
    $y_i^\sigma := \mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, \mathsf{k}_\sigma, x_i)$
    $y_i^{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_i^\sigma)$
$b \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, (x_i, y_i^{1-\sigma})_{i\in[N]})$
return $b$
```

Fig. 13: Security of game for a weak PK-PCF. Here, $\mathsf{RSample}$ is as defined in Definition 12.

Definition 13 (Public-Key Pseudorandom Correlation Function [BCM+24])**.** *Let λ be a security parameter, $\mathcal{Y}$ be a reverse-sampleable correlation with output length $n = n(\lambda) \in \mathsf{poly}(\lambda)$, and $\lambda \leq m = m(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. A Public-Key Pseudorandom Correlation Function (PK-PCF) for $\mathcal{Y}$ is defined by a tuple of algorithms $\mathsf{pkPCF} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{KeyDer}, \mathsf{Eval})$ with the following functionality:*

- $\mathsf{pkPCF.Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter λ and outputs a common reference string (CRS) crs.*
- $\mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$. *The randomized key generation algorithm takes as input the CRS crs and a party identifier $\sigma \in \{0,1\}$. It outputs a public and secret key pair $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$ for the party.*
- $\mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{pk}_{1-\sigma}, \mathsf{sk}_\sigma) \to \mathsf{k}_\sigma$. *The deterministic key derivation algorithm takes as input the CRS crs, a party identifier $\sigma \in \{0,1\}$, the public key $\mathsf{pk}_{1-\sigma}$ of another party, and the secret key sk_σ of this party. It outputs an evaluation key k_σ for this party.*
- $\mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, \mathsf{k}^\sigma, x) \to y_\sigma$. *The deterministic evaluation algorithm takes as input the CRS, the party identifier $\sigma \in \{0,1\}$, an evaluation key k^σ, and an input $x \in \{0,1\}^m$. It outputs a string $y^\sigma \in \{0,1\}^n$.*

We say that $\mathsf{pkPCF} = (\mathsf{KeyGen}, \mathsf{Eval})$ *is a PK-PCF for the reverse-sampleable correlation* $\mathcal{Y}$*, if the following two properties hold:*

Correctness / Pseudorandom $\mathcal{Y}$*-correlated outputs. For every* $\sigma \in \{0,1\}$*, all efficient adversaries* $\mathcal{A}$*, and all* $N = N(\lambda) \in \mathsf{poly}(\lambda)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\mathsf{Adv}^{\mathsf{pr}}_{\mathcal{A},N}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$*, for* $b \in \{0,1\}$*, is as defined in Figure 12. In particular, the adversary is given access to* N *samples.*

Security. For all $\sigma \in \{0,1\}$*, and all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{A},N,\sigma}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,b}(\lambda)$*, for* $b \in \{0,1\}$*, is as defined in Figure 13 (again, with the adversary given* N *samples).*

Remark 6 (Weak vs. Strong PCFs). We remark that, contrary to pseudorandom functions, the default notion of a PCF is a *weak* PCF, where the inputs are chosen uniformly at random. A PCF (as defined in Definition 13) can be generically converted to a strong PCF using a random oracle.

6.2 Public-key PCFs from MKHSS

In this section, we provide a construction of a PK-PCF for any additive correlations computable by RMS programs (which includes the class NC^1). We start by defining additive correlations:

Definition 14 (Additive Correlation)**.** *Let* λ *be a security parameter, and let* $n_y = n_y(\lambda) \in \mathsf{poly}(\lambda)$ *be an input length and* $n_z = n_z(\lambda) \in \mathsf{poly}(\lambda)$ *be an output length. We say that* $\mathcal{Y}$ *(as defined in Definition 12) is an* additive correlation *over a ring* $\mathcal{R}$ *defined by a function family* $\{C_\lambda \colon \mathcal{R}^{n_y} \times \mathcal{R}^{n_y} \to \mathcal{R}^{n_z}\}_{\lambda\in\mathbb{N}}$ *if* $\mathcal{Y}(1^\lambda)$ *outputs pairs of samples* $((y_0, z_0), (y_1, z_1))$*, where* $(y_\sigma, z_\sigma) \in \mathcal{R}^{n_y} \times \mathcal{R}^{n_z}$ *are uniformly random conditioned on* $z_0 + z_1 = C_\lambda(y_0, y_1)$ *We will drop the subscript* λ *when clear from context.*

Remark 7. Additive correlations are naturally reverse-samplable. To see this, observe that given (y_σ, z_σ), it is possible to efficiently sample $y_{1-\sigma} \leftarrow \mathcal{R}^{n_y}$, and set $z_{1-\sigma} := C(y_0, y_1) - z_\sigma$.

We present our PK-PCF construction for correlations computable by RMS programs in Figure 14.

6.2.1 Security analysis

We now turn to the security analysis of the PK-PCF from Figure 14.

Theorem 4 (Security of PK-PCF)**.** *Assuming the existence of an externally-secure MKHSS scheme* MKHSS *for polynomial-size RMS programs over a finite ring* $\mathcal{R}$ *and the existence of PRFs in* NC^1*, the construction described in Figure 14 is a PK-PCF for arbitrary additive correlations described by polynomial-size RMS programs over* $\mathcal{R}$*.*

Pseudorandomness. Consider the following sequence of hybrid games.

– *Hybrid* $\mathcal{H}_0$. This hybrid game consists of the pseudorandomness experiment $\mathsf{E}^{\mathsf{pkpr}}_{\mathcal{A},N,0}$.

– *Hybrid* $\mathcal{H}_1$. In this hybrid game, the outputs (z_0, z_1) are computed by first sampling $z_1 \leftarrow\$\, \mathcal{R}$, and then setting $z_0 := z_1 + P_{\mathbf{x}}(k_0, k_1)$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$ assuming the external security of MKHSS.

  Proof. An efficient distinguisher immediately contradicts the external security of MKHSS. □

– *Hybrid* $\mathcal{H}_2$. In this hybrid game, the challenger is given oracle access to $F_{k_\sigma}(\cdot)$, for all $\sigma \in \{0,1\}$. Instead of computing F_{k_σ} using k_σ, the challenger obtains the PRF evaluation by querying the respective oracles. Then, the challenger samples $z_1 \leftarrow\$\, \mathcal{R}$, and set $z_0 := z_1 + C(\mathbf{y}_0, \mathbf{y}_1)$.

Public-Key PCF from MKHSS

Public Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be an externally secure MKHSS scheme for polynomial-size RMS programs defined over a ring $\mathcal{R}$. Let $n_k = n_k(\lambda)$ and $n_x = n_x(\lambda)$ be polynomials denoting the PRF key length and output length, respectively. Let $F_k \colon \mathcal{R}^m \to \mathcal{R}^{n_x}$ be a PRF with keys sampled from $\mathcal{R}^{n_k}$, such that $F_k(\mathbf{x})$ is computable by a polynomial-size RMS program over $\mathcal{R}$. Let $\mathcal{Y}$ be an additive correlation over a ring $\mathcal{R}$ defined by a correlation circuit C computable by polynomial-size RMS programs.

The evaluated program. For all vectors $\mathbf{x} \in \mathcal{R}^m$, define the program $P_{\mathbf{x}} \colon \mathcal{R}^{n_k} \times \mathcal{R}^{n_k} \to \mathcal{R}$ to be the polynomial-size RMS program that, on input $k_0, k_1 \in \mathcal{R}^{n_k}$, computes $\mathbf{y}_\sigma := F_{k_\sigma}(\mathbf{x}) \in \mathcal{R}^{n_x}$, for all $\sigma \in \{0,1\}$, and outputs $C(\mathbf{y}_0, \mathbf{y}_1)$.

```
$\mathsf{pkPCF.Setup}(1^\lambda)$:
  1 : $\mathsf{crs} \leftarrow \mathsf{MKHSS.Setup}(\lambda)$
  2 : return $\mathsf{crs}$

$\mathsf{pkPCF.KeyGen}(\mathsf{crs}, \sigma)$:
  1 : $k \leftarrow \mathcal{R}^{n_k}$
  2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$
  3 : $([\![k]\!]_0^\sigma, [\![k]\!]_1^\sigma) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, k)$
  4 : $\mathsf{pk}_\sigma^{\mathsf{pcf}} := (\mathsf{pk}, [\![k]\!]_{1-\sigma}^\sigma)$
  5 : $\mathsf{sk}_\sigma^{\mathsf{pcf}} := (\mathsf{sk}, [\![k]\!]_\sigma^\sigma, k)$
  6 : return $(\mathsf{pk}_\sigma^{\mathsf{pcf}}, \mathsf{sk}_\sigma^{\mathsf{pcf}})$

$\mathsf{pkPCF.KeyDer}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma^{\mathsf{pcf}}, \mathsf{pk}_{1-\sigma}^{\mathsf{pcf}})$:
  1 : return $\mathsf{k}_\sigma := (\mathsf{sk}_\sigma^{\mathsf{pcf}}, \mathsf{pk}_{1-\sigma}^{\mathsf{pcf}})$

$\mathsf{pkPCF.Eval}(\mathsf{crs}, \sigma, \mathsf{k}_\sigma, \mathbf{x})$:
  1 : parse $\mathsf{k}_\sigma := ((\mathsf{sk}_\sigma, [\![k_\sigma]\!]_\sigma^\sigma, k_\sigma), (\mathsf{pk}_{1-\sigma}, [\![k_{1-\sigma}]\!]_\sigma^{1-\sigma}))$
  2 : $\mathbf{y}_\sigma := F_{k_\sigma}(\mathbf{x})$
  3 : $z_\sigma := \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![k_\sigma]\!]_\sigma^\sigma, [\![k_{1-\sigma}]\!]_\sigma^{1-\sigma}, P_{\mathbf{x}})$
  4 : return $(\mathbf{y}_\sigma, z_\sigma)$
```

Fig. 14: Public-key PCF from MKHSS.

Claim. $\mathcal{H}_2 \approx_s \mathcal{H}_1$.

Proof. By definition of $P_{\mathbf{x}}$ in Figure 14, the output distribution is identical to that of $\mathcal{H}_1$. ☐

– *Hybrid* $\mathcal{H}_3$. This hybrid game proceeds as $\mathcal{H}_2$, except that the key k in $\mathsf{pkPCF.KeyGen}$ is replaced by 0. That is, $\mathsf{pkPCF.KeyGen}$ computes $([\![k]\!]_0^\sigma, [\![k]\!]_1^\sigma) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, 0)$.

  Claim. $\mathcal{H}_3 \approx_c \mathcal{H}_2$ assuming the security of MKHSS.

  Proof. The claim follows immediately by the security of MKHSS. ☐

– *Hybrid* $\mathcal{H}_4$. This hybrid game proceeds as $\mathcal{H}_3$, except that the PRF oracles F_{k_σ}, for $\sigma \in \{0,1\}$, are replaced with *random* oracles H_σ. Hence, $\mathbf{y}_\sigma$ is computed as $\mathbf{y}_\sigma := H_\sigma(\mathbf{x})$, for all $\sigma \in \{0,1\}$.

  Claim. $\mathcal{H}_4 \approx_c \mathcal{H}_3$ assuming the security of the PRF.

  Proof. The claim follows from the standard PRF security property (note that we can apply the PRF security since the shares $([\![k_\sigma]\!]_0^\sigma, [\![k_\sigma]\!]_1^\sigma)$ do not depend on the PRF key k_σ anymore). ☐

– *Hybrid* $\mathcal{H}_5$. In this hybrid game, the challenger samples $\mathbf{y}_\sigma \leftarrow\$\, \mathcal{R}^{n_y}$, for all $\sigma \in \{0,1\}$.

Claim. $\mathcal{H}_5 \approx_s \mathcal{H}_3$.

Proof. Observe that the probability of any two inputs $\mathbf{x}$ to the random oracle H_σ colliding is negligible, hence $\mathcal{H}_5$ is statistically indistinguishable from $\mathcal{H}_4$. □

At this point, it suffices to note that $\mathcal{H}_5$ is equivalent to the experiment $\mathsf{E}^{\mathsf{pkpr}}_{\mathcal{A},N,1}$, concluding the proof.

Security. We now turn to proving the security of our PK-PCF. We proceed as above via a sequence of hybrid games.

– *Hybrid* $\mathcal{H}_0$. This hybrid game consists of the security experiment $\mathsf{E}^{\mathsf{pksec}}_{\mathcal{A},N,0}$.

– *Hybrid* $\mathcal{H}_1$. In this hybrid game, the challenger computes $z_{1-\sigma}$ by first computing

$$z_\sigma \leftarrow \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![k_\sigma]\!]^\sigma_\sigma, [\![k_{1-\sigma}]\!]^{1-\sigma}_\sigma, C_\mathbf{x})$$

and then setting $z_{1-\sigma} := z_\sigma + (-1)^{1-\sigma} C_\mathbf{x}(\mathbf{y}_0, \mathbf{y}_1)$.

Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$ assuming the correctness of MKHSS.

Proof. The claim follows directly from the correctness property of MKHSS. □

Hybrid $\mathcal{H}_2$. This hybrid game is identical to $\mathcal{H}_1$ except that $\mathsf{pkPCF.KeyGen}$ outputs:

$$([\![k_{1-\sigma}]\!]^{1-\sigma}_0, [\![k_{1-\sigma}]\!]^{1-\sigma}_1) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, 1-\sigma, \mathsf{pk}, 0).$$

Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ assuming the security of MKHSS.

Proof. The claim follows directly from the security property of MKHSS. □

– *Hybrid* $\mathcal{H}_3$. In this hybrid game, the challenges is given oracle access to $F_{k_{1-\sigma}}$. The challenger uses this oracle access to compute $\mathbf{y}_{1-\sigma} := F_{k_{1-\sigma}}(\mathbf{x}) \in \mathcal{R}^{n_x}$.

Claim. $\mathcal{H}_3 \approx_s \mathcal{H}_2$.

Proof. By definition of $P_\mathbf{x}$, the output distribution is identical to that of $\mathcal{H}_2$. □

– *Hybrid* $\mathcal{H}_4$. This hybrid game is identical to $\mathcal{H}_3$ except that the oracle for $F_{k_{1-\sigma}}$ is now replaced with a random oracle H. Hence, $\mathbf{y}_{1-\sigma}$ is computed as $\mathbf{y}_{1-\sigma} := H(\mathbf{x})$.

Claim. $\mathcal{H}_4 \approx_c \mathcal{H}_3$ assuming the security of the PRF.

Proof. The claim follows from the security of the PRF. In particular, note that we can apply the PRF security since the shares $([\![k_{1-\sigma}]\!]^{1-\sigma}_0, [\![k_{1-\sigma}]\!]^{1-\sigma}_1)$ do not depend on $k_{1-\sigma}$ anymore. □

– *Hybrid* $\mathcal{H}_5$. In this hybrid game, the challenger samples $\mathbf{y}_{1-\sigma} \leftarrow\$ \mathcal{R}^{n_y}$.

Claim. $\mathcal{H}_5 \approx_s \mathcal{H}_4$.

Proof. Observe that the probability of any two inputs $\mathbf{x}$ to H colliding is negligible, hence this hybrid is statistically indistinguishable from $\mathcal{H}_4$. □

At this point, it suffices to note that $\mathsf{E}^{\mathsf{pksec}}_{\mathcal{A},N,1}$ uses the natural reverse-sampling algorithm for additive correlations. This concludes the proof.

Remark 8 (On strong PK-PCFs). We note that because we use a standard PRF in our construction, our PK-PCF can be shown to be a *strong* PCF. Alternatively, we can substitute the PRF for a *weak* PRF and follow the same proof.

```
$\underline{\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,0}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{mpkPCF.Setup}(1^\lambda)$
$(x_1, \ldots, x_N) \leftarrow (\{0,1\}^n)^N$
foreach $i \in [p]$:
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$
  foreach $j \in [N]$:
      $y_{i,j} \leftarrow \mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_\ell)_{\ell \neq i}, x_j)$
$b \leftarrow \mathcal{A}((\mathsf{pk}_1, \ldots, \mathsf{pk}_p), (x_1, \ldots, x_N), (y_{i,j})_{i \leq p, j \leq N})$
return $b$
```

```
$\underline{\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,1}(\lambda)}$:
$\mathsf{crs} \leftarrow \mathsf{mpkPCF.Setup}(1^\lambda)$
$(x_1, \ldots, x_N) \leftarrow (\{0,1\}^n)^N$
foreach $i \in [p]$:
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$
foreach $j \in [N]$:
  $(y_{1,j}, \ldots, y_{p,j}) \leftarrow \mathcal{Y}(1^\lambda)$
$b \leftarrow \mathcal{A}((\mathsf{pk}_1, \ldots, \mathsf{pk}_p), (x_1, \ldots, x_N), (y_{i,j})_{i \leq p, j \leq N})$
return $b$
```

Fig. 15: Pseudorandomness of a multi-party public-key PCF for a p-party correlation $\mathcal{Y}$.

6.3 Multi-party computation with silent preprocessing

Building upon the PK-PCF introduced in Section 6.2, we introduce a *multi-party* PK-PCF for generating Beaver triple correlations, and discuss the direct implications to secure computation. We note that we can only support degree-2 correlations (e.g., Beaver triples) in the multi-party setting when using two-party PCFs as a building block. The same limitation applies to prior constructions of multi-party correlation generators from two-party building blocks [BCG$^+$19b].

Defining multi-party PK-PCFs. We start by introducing the notion of multi-party PK-PCF (Definition 15). Our definition generalizes the notion of PK-PCF to more than two parties in a natural way. Note that for simplicity, we "absorb" the key derivation procedure into $\mathsf{MKHSS.Eval}$. That is, in our formal definition, $\mathsf{MKHSS.Eval}$ directly takes as input the secret key sk_i of a party and the public keys $(\mathsf{pk}_j)_{j \neq i}$ of the other parties. This is without loss of generality, as we can always define KeyDer to output $\mathsf{k}_i := (\mathsf{sk}_i, (\mathsf{pk}_j)_{j \neq i})$. Indeed, we note that this is exactly what our MKHSS-based PK-PCF construction in Figure 14 does.

Definition 15 (Multi-Party Public-Key Pseudorandom Correlation Function)**.** *A multi-party PK-PCF for a p-party correlation $\mathcal{Y}$ is defined by a tuple of algorithms* $\mathsf{mpkPCF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval})$*, with the following template:*

– $\mathsf{mpkPCF.Setup}(1^\lambda) \to \mathsf{crs}$*. The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs*.*
– $\mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$*. The randomized key generation algorithm takes as input the CRS and an index $i \in [p]$, outputs a pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ of public and private* mpkPCF *keys.*
– $\mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_j)_{j \neq i}, x) \to y_i$*. The deterministic evaluation algorithm takes as input an index i, the secret key sk_i, the public keys $(\mathsf{pk}_j)_{j \neq i}$, and an input $x \in \{0,1\}^n$. It outputs a string y_i.*

A multi-party PK-PCF must satisfy the following pseudorandomness *and* security *properties:*

Correctness / Pseudorandom $\mathcal{Y}$-correlated Outputs. For all efficient adversaries $\mathcal{A}$, and all $N = N(\lambda) \in \mathsf{poly}(\lambda)$, there exists a negligible function negl *such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{mpkpr}}_{\mathcal{A},N}(\lambda) := \left| \Pr[\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,1}(\lambda) = 1] \right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 15.

Security. There exists an efficient algorithm $\mathsf{RSample}\colon (1^\lambda, i^*, (y_i)_{i \neq i^*}) \mapsto y_{i^*}$ *such that for every efficient adversary $\mathcal{A}$, $N = N(\lambda) \in \mathsf{poly}(\lambda)$, and every $i^* \in [p]$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\mathsf{Adv}^{\mathsf{mpkpr}}_{\mathcal{A},N}(\lambda, i^*) := \left| \Pr[\mathsf{E}^{\mathsf{mpkpr}}_{\mathcal{A},N,0}(\lambda, i^*) = 1] - \Pr[\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,1}(\lambda, i^*) = 1] \right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,b}(\lambda, i^*)$, for $b \in \{0,1\}$, is as defined in Figure 16.

```
$\underline{\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,0}(\lambda, i^*)}$:
$\mathsf{crs} \leftarrow \mathsf{mpkPCF.Setup}(1^\lambda)$
$(x_1, \ldots, x_N) \leftarrow (\{0,1\}^n)^N$
foreach $i \in [p]$:
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$
    foreach $j \in [N]$:
        $y_{i,j} \leftarrow \mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_\ell)_{\ell \neq i}, x_j)$
$b \leftarrow \mathcal{A}((\mathsf{pk}_1, \ldots, \mathsf{pk}_p), \mathsf{sk}_{i^*}, (x_j)_{j \leq N}, (y_{i,j})_{i \neq i^*, j \leq N})$
return $b$
```

```
$\underline{\mathsf{E}^{\mathsf{mpksec}}_{\mathcal{A},N,1}(\lambda, i^*)}$:
$\mathsf{crs} \leftarrow \mathsf{mpkPCF.Setup}(1^\lambda)$
$(x_1, \ldots, x_N) \leftarrow (\{0,1\}^n)^N$
foreach $i \in [p]$:
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$
    foreach $j \in [N]$:
        $y_{i,j} \leftarrow \mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_\ell)_{\ell \neq i}, x_j)$
        $y_{i^*,j} \leftarrow \mathsf{RSample}(1^\lambda, i^*, (y_{i,j})_{i \neq i^*})$
$b \leftarrow \mathcal{A}((\mathsf{pk}_1, \ldots, \mathsf{pk}_p), \mathsf{sk}_{i^*}, (x_j)_{j \leq N}, (y_{i,j})_{i \neq i^*, j \leq N})$
return $b$
```

Fig. 16: Security of a multi-party public-key PCF for a p-party correlation $\mathcal{Y}$.

Construction. We construct a multi-party PK-PCF for the p-party Beaver triple correlation over a ring $\mathcal{R}$. Let $\mathcal{B}$ denote the p-party correlation that, on input λ, samples p uniformly random triples $(a_i, b_i, c_i) \leftarrow\!\$\, \mathcal{R}^3$ conditioned on $(\sum_i a_i) \cdot (\sum_i b_i) = \sum_i c_i$. We represent our construction on Figure 17.

At a high level, the construction of Figure 17 is a direct extension of the PK-PCF construction from Figure 14. In particular, the generalization from two parties to p parties is fairly straightforward. In a little more detail, our multi-party PK-PCF is realized as follows:

- Each party P_i generates an MKHSS keys pair $(\mathsf{pk}_i^{\mathsf{mkhss}}, \mathsf{sk}_i^{\mathsf{mkhss}})$, samples a PRF key k_i, and shares k_i into $([\![k]\!]_0^\sigma, [\![k]\!]_1^\sigma)$ using $\mathsf{MKHSS.Share}$, for each $\sigma \in \{0,1\}$. Then, P_i sets: $\mathsf{sk}_i := (\mathsf{sk}, [\![k]\!]_1^0, [\![k]\!]_1^1, k)$ and $\mathsf{pk}_i := (\mathsf{pk}, [\![k]\!]_0^0, [\![k]\!]_0^1)$.
- On input $\mathbf{x}$, each party P_i defines $(a_i, b_i) := F_{k_i}(\mathbf{x})$.
- Finally, each pair of parties P_i, P_j, using their MKHSS shares of k_i and k_j, computes additive shares of $F_{k_i}(\mathbf{x}) \cdot F_{k_j}(\mathbf{x})$. Each party P_i aggregates all the shares computed in this way into c_i.

By correctness of the MKHSS scheme, it holds that:

$$\sum_i c_i = \sum_{i,j} F_{k_i}(\mathbf{x}) \cdot F_{k_j}(\mathbf{x}) = \left(\sum_i a_i\right) \cdot \left(\sum_i b_i\right).$$

Theorem 5 (Security of multi-party PK-PCF)**.** *Assuming the existence of an externally-secure MKHSS scheme* MKHSS *for polynomial-size RMS programs and a PRF in* NC^1*, the construction described in Figure 17 is a multi-party PK-PCF for Beaver triple correlations.*

Proof (sketch). The proof is essentially identical to the proof of Theorem 4. ■

Application to secure computation. A multi-party PK-PCF for the p-party Beaver triple correlation immediately implies a p-party semi-honest secure computation protocol for a general arithmetic circuit C over $\mathcal{R}$ in the *silent preprocessing model* (see Boyle et al. [BCGI18, BCG+19a, BCG+19b, BCG+20a] for discussions on this model):

Preprocessing phase. Each party P_i runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mpkPCF.KeyGen}(i)$ and broadcasts pk_i.

Silent expansion. For each multiplication gate in C, each party P_i computes $(a_i, b_i, c_i) := \mathsf{pkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_j)_{j \neq i}, \mathbf{x})$, where $\mathbf{x}$ is a fresh common randomness.

Online phase. The parties run the information-theoretic GMW protocol, consuming one Beaver triple for each multiplication gate computed in the preprocessing phase.

The fact that GMW can be securely instantiated using the correlated pseudorandomness generated by a (multi-party, public key) PCF follows from the fact that the latter suffices to instantiate a *corruptible* functionality for generating correlated randomness, and GMW is provably secure given ideal access to a corruptible correlated randomness functionality. We refer the reader to Boyle et al. [BCG+19b] for more detailed discussion about this approach. Then, plugging in our construction of (statistically correct) MKHSS from DCR or class group assumptions, we get the following corollary:

Corollary 4. *Assume either the DCR assumption or the DDH assumption over class groups. For any polynomial number of parties p, for any polynomial-size arithmetic circuit C with n inputs, s*

Multi-Party Public-key PCF from MKHSS

Public Parameters. Let $\mathsf{MKHSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Share}, \mathsf{Eval})$ be an MKHSS scheme for polynomial-size RMS programs defined over a ring $\mathcal{R}$. Let $F_k : \mathcal{R}^m \to \mathcal{R}^2$ be a PRF with keys sampled from $\mathcal{R}^{n_k}$, such that $F_k(\mathbf{x})$ is computable by a polynomial-size RMS program over $\mathcal{R}$.

The evaluated program. For all vectors $\mathbf{x} \in \mathcal{R}^m$, define $P_{\mathbf{x}} \colon \mathcal{R}^{n_k} \times \mathcal{R}^{n_k} \to \mathcal{R}$ to be the function that, on input $k_0, k_1 \in \mathcal{R}^{n_k}$, computes $(a_\sigma, b_\sigma) := F_{k_\sigma}(\mathbf{x})$, for all $\sigma \in \{0,1\}$, and outputs $a_0 b_1 + a_1 b_0$.

```
$\mathsf{mpkPCF.Setup}(1^\lambda)$:
1 : $\mathsf{crs} \leftarrow \mathsf{MKHSS.Setup}(\lambda)$
2 : return $\mathsf{crs}$
```

```
$\mathsf{mpkPCF.KeyGen}(\mathsf{crs}, i)$:
1 : $k \leftarrow\$ \mathcal{R}^{n_k}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MKHSS.KeyGen}(\mathsf{crs})$
3 : foreach $\sigma \in \{0,1\}$:
4 :     $(\llbracket k \rrbracket_0^\sigma, \llbracket k \rrbracket_1^\sigma) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}, k)$
5 : $\mathsf{pk}_i := (\mathsf{pk}, \llbracket k \rrbracket_0^0, \llbracket k \rrbracket_0^1)$
6 : $\mathsf{sk}_i := (\mathsf{sk}, \llbracket k \rrbracket_1^0, \llbracket k \rrbracket_1^1, k)$
7 : return $(\mathsf{pk}_i, \mathsf{sk}_i)$
```

```
$\mathsf{mpkPCF.Eval}(\mathsf{crs}, i, \mathsf{sk}_i, (\mathsf{pk}_j)_{j \neq i}, \mathbf{x})$:
1 : $(a_i, b_i) := F_{k_i}(\mathbf{x})$
2 : parse $\mathsf{sk}_i = (\mathsf{sk}, \llbracket k_i \rrbracket_1^0, \llbracket k_i \rrbracket_1^1, k_i)$
3 : $c_i := a_i \cdot b_i$
4 : foreach $j \in [p] \setminus \{i\}$:
5 :     parse $\mathsf{pk}_j = (\mathsf{pk}, \llbracket k_j \rrbracket_0^0, \llbracket k_j \rrbracket_0^1)$
6 :     if $j > i$ then $\sigma := 0$   else $\sigma := 1$
7 :     $c_i := c_i + \mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_i, \mathsf{pk}_j, \llbracket k_i \rrbracket_1^\sigma, \llbracket k_j \rrbracket_0^{1-\sigma}, P_{\mathbf{x}})$
8 : return $(a_i, b_i, c_i)$
```

Fig. 17: Multi-party Public-key PCF.

multiplication gates, and m outputs over a ring $\mathcal{R}$, there exists a p-party protocol securely computing C in the preprocessing model against an adversary passively corrupting up to $p-1$ parties with the following communication:

- *In the preprocessing phase, the parties communicate $p \cdot \mathsf{poly}(\lambda)$ bits in a single round of broadcast.*
- *In the online phase, the parties communicate $p \cdot (2s+m)$ elements of $\mathcal{R}$.*

Previously, the best-known multi-party protocols with silent preprocessing (under assumptions not known to imply spooky encryption) were constructed using either HSS (from DCR or DDH over class groups [OSY21, RS21, ADOS22]), programmable 2-party PCGs (from ring-LPN [BCG⁺20b], or quasi-abelian syndrome decoding [BCCD23]). All these approaches incurred a *quadratic* communication overhead $\tilde{\Omega}(p^2) \cdot \mathsf{poly}(\lambda)$ in the number of parties p, in the preprocessing phase. Our construction is the first to achieve $p \cdot \mathsf{poly}(\lambda)$ communication overhead in the preprocessing phase, which is quasi-optimal.

Acknowledgments

We thank Srini Devadas for helpful comments on the introduction. We thank the anonymous reviewers for helpful comments and suggestions that improved the presentation and results in the paper. Geoffroy Couteau and Aditya Hegde acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE) and under the France 2030 ANR Project ANR-22-PECY-003 SecureCompute. This work is supported by ERC grant OBELiSC (101115790). Aditya Hegde and Abhishek Jain were supported by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. Abhishek Jain was additionally supported by JP Morgan Faculty Award, and research gifts from Ethereum, Stellar, and Cisco.

Bibliography

[ADOS22] D. Abram, I. Damgård, C. Orlandi, and P. Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 421–452. Springer, Cham, August 2022.

[AJJM20] P. Ananth, A. Jain, Z. Jin, and G. Malavolta. Multi-key fully-homomorphic encryption in the plain model. In *TCC 2020, Part I*, *LNCS* 12550, pages 28–57. Springer, Cham, November 2020.

[AJJM21] P. Ananth, A. Jain, Z. Jin, and G. Malavolta. Unbounded multi-party computation from learning with errors. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 754–781. Springer, Cham, October 2021.

[AJL$^+$12] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT 2012*, *LNCS* 7237, pages 483–501. Springer, Berlin, Heidelberg, April 2012.

[ARS24] D. Abram, L. Roy, and P. Scholl. Succinct homomorphic secret sharing. In *EUROCRYPT 2024, Part VI*, *LNCS* 14656, pages 301–330. Springer, Cham, May 2024.

[BBC$^+$24] M. Bombar, D. Bui, G. Couteau, A. Couvreur, C. Ducros, and S. Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$OLE-based multi-party computation for boolean circuits. In *ASIACRYPT 2024, Part VI, LNCS 15489*, pages 69–101. Springer, Singapore, 2024.

[BCCD23] M. Bombar, G. Couteau, A. Couvreur, and C. Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In *CRYPTO 2023, Part IV*, *LNCS* 14084, pages 567–601. Springer, Cham, August 2023.

[BCCS24] A. Bondarchuk, O. Chakraborty, G. Couteau, and R. Sirdey. Downlink (t)FHE ciphertexts compression. Cryptology ePrint Archive, Paper 2024/1921, 2024.

[BCG$^+$17] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

[BCG$^+$19a] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG$^+$19b] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 489–518. Springer, Cham, August 2019.

[BCG$^+$20a] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

[BCG$^+$20b] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *CRYPTO 2020, Part II*, *LNCS* 12171, pages 387–416. Springer, Cham, August 2020.

[BCGI18] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BCM23] E. Boyle, G. Couteau, and P. Meyer. Sublinear-communication secure multiparty computation does not require FHE. In *EUROCRYPT 2023, Part II*, *LNCS* 14005, pages 159–189. Springer, Cham, April 2023.

[BCM$^+$24] D. Bui, G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. Fast public-key silent OT and more from constrained Naor-Reingold. In *EUROCRYPT 2024, Part VI*, *LNCS* 14656, pages 88–118. Springer, Cham, May 2024.

[BCP03] E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *ASIACRYPT 2003*, *LNCS* 2894, pages 37–54. Springer, Berlin, Heidelberg, November / December 2003.

[BDSS25] E. Boyle, L. Devadas, and S. Servan-Schreiber. Non-interactive distributed point functions. Cryptology ePrint Archive, Paper 2025/095, 2025.

[Bea92] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO'91*, *LNCS* 576, pages 420–432. Springer, Berlin, Heidelberg, August 1992.

[Bea95] D. Beaver. Precomputing oblivious transfer. In *CRYPTO'95*, *LNCS* 963, pages 97–109. Springer, Berlin, Heidelberg, August 1995.

[BGI16] E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, *LNCS* 9814, pages 509–539. Springer, Berlin, Heidelberg, August 2016.

[BGI17] E. Boyle, N. Gilboa, and Y. Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In *EUROCRYPT 2017, Part II*, *LNCS* 10211, pages 163–193. Springer, Cham, April / May 2017.

[BGI$^+$18] E. Boyle, N. Gilboa, Y. Ishai, H. Lin, and S. Tessaro. Foundations of homomorphic secret sharing. In *ITCS 2018*, pages 21:1–21:21. LIPIcs, January 2018.

[BGMM20] J. Bartusek, S. Garg, D. Masny, and P. Mukherjee. Reusable two-round MPC from DDH. In *TCC 2020, Part II*, *LNCS* 12551, pages 320–348. Springer, Cham, November 2020.

[BHHO08] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In *CRYPTO 2008*, *LNCS* 5157, pages 108–125. Springer, Berlin, Heidelberg, August 2008.

[BJKL21] F. Benhamouda, A. Jain, I. Komargodski, and H. Lin. Multiparty reusable non-interactive secure computation from LWE. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 724–753. Springer, Cham, October 2021.

[BKS19] E. Boyle, L. Kohl, and P. Scholl. Homomorphic secret sharing from lattices without FHE. In *EUROCRYPT 2019, Part II*, *LNCS* 11477, pages 3–33. Springer, Cham, May 2019.

[BL20] F. Benhamouda and H. Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In *TCC 2020, Part II*, *LNCS* 12551, pages 349–378. Springer, Cham, November 2020.

[BM90] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *CRYPTO'89*, *LNCS* 435, pages 547–557. Springer, New York, August 1990.

[CDD$^+$24] G. Couteau, L. Devadas, S. Devadas, A. Koch, and S. Servan-Schreiber. QuietOT: Lightweight oblivious transfer with a public-key setup. In *ASIACRYPT 2024, Part II, LNCS 15485*, pages 197–231. Springer, Singapore, 2024.

[CK24] G. Couteau and N. Kumar. 10-party sublinear secure computation from standard assumptions. In *CRYPTO 2024, Part IX*, *LNCS* 14928, pages 39–73. Springer, Cham, August 2024.

[CKS08] D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In *EUROCRYPT 2008*, *LNCS* 4965, pages 127–145. Springer, Berlin, Heidelberg, April 2008.

[Cle90] R. Cleve. Towards optimal simulations of formulas by bounded-width programs. In *22nd ACM STOC*, pages 271–277. ACM Press, May 1990.

[CM21] G. Couteau and P. Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 842–870. Springer, Cham, October 2021.

[CMPR23] G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In *EUROCRYPT 2023, Part III*, *LNCS* 14006, pages 194–224. Springer, Cham, April 2023.

[Cou19] G. Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In *EUROCRYPT 2019, Part II*, *LNCS* 11477, pages 473–503. Springer, Cham, May 2019.

[CS02] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT 2002*, *LNCS* 2332, pages 45–64. Springer, Berlin, Heidelberg, April / May 2002.

[CZ22] G. Couteau and M. Zarezadeh. Non-interactive secure computation of inner-product from LPN and LWE. In *ASIACRYPT 2022, Part I*, *LNCS* 13791, pages 474–503. Springer, Cham, December 2022.

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

[DHRW16] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 93–122. Springer, Berlin, Heidelberg, August 2016.

[DIJL23] Q. Dao, Y. Ishai, A. Jain, and H. Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In *CRYPTO 2023, Part II*, *LNCS* 14082, pages 315–348. Springer, Cham, August 2023.

[DJ03] I. Damgård and M. Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP 03*, *LNCS* 2727, pages 350–364. Springer, Berlin, Heidelberg, July 2003.

[DKK18] I. Dinur, N. Keller, and O. Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In *CRYPTO 2018, Part III*, *LNCS* 10993, pages 213–242. Springer, Cham, August 2018.

[DPSZ12] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, *LNCS* 7417, pages 643–662. Springer, Berlin, Heidelberg, August 2012.

[FHKP13] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange. In *PKC 2013*, *LNCS* 7778, pages 254–271. Springer, Berlin, Heidelberg, February / March 2013.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GI14] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, *LNCS* 8441, pages 640–658. Springer, Berlin, Heidelberg, May 2014.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[Gol06] O. Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA, 2006.

[HLP11] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO 2011*, *LNCS* 6841, pages 132–150. Springer, Berlin, Heidelberg, August 2011.

[JRX24] J. Januzelli, L. Roy, and J. Xu. Under what conditions is encrypted key exchange actually secure? Cryptology ePrint Archive, Report 2024/324, 2024.

[KKL$^+$16] V. Kolesnikov, H. Krawczyk, Y. Lindell, A. J. Malozemoff, and T. Rabin. Attribute-based key exchange with general policies. In *ACM CCS 2016*, pages 1451–1463. ACM Press, October 2016.

[LTV12] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.

[Mel22] K. Melissaris. *Witness-Authenticated Key Exchange*. PhD thesis, City University of New York, 2022.

[MW16] P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 735–763. Springer, Berlin, Heidelberg, May 2016.

[NS09] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO 2009*, *LNCS* 5677, pages 18–35. Springer, Berlin, Heidelberg, August 2009.

[OSY21] C. Orlandi, P. Scholl, and S. Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, *LNCS* 12696, pages 678–708. Springer, Cham, October 2021.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, *LNCS* 1592, pages 223–238. Springer, Berlin, Heidelberg, May 1999.

[PS16] C. Peikert and S. Shiehian. Multi-key FHE from LWE, revisited. In *TCC 2016-B, Part II*, *LNCS* 9986, pages 217–238. Springer, Berlin, Heidelberg, October / November 2016.

[RS21] L. Roy and J. Singh. Large message homomorphic secret sharing from DCR and applications. In *CRYPTO 2021, Part III*, *LNCS* 12827, pages 687–717, Virtual Event, August 2021. Springer, Cham.

[XW23] P. Xu and L.-P. Wang. Multi-key homomorphic secret sharing from LWE without multi-key HE. In *ACISP 23*, *LNCS* 13915, pages 248–269. Springer, Cham, July 2023.

Supplementary Material

A Alternative Construction of Multi-Key HSS from DCR

In this section, we provide an alternative MKHSS for NC^1 computations from DCR. This alternative construction has the benefit of not needing the DDH assumption over the Paillier group and avoiding the short exponent assumption, and providing a different path to realizing MKHSS. In particular, unlike our construction in Section 4, this alternative construction does *not* require "expanding" the input shares and secret keys. Moreover, it provides a path to an implementable (albeit concretely slow) MKHSS construction, as we explain later. We note that the lack of ciphertext and key expansion is potentially of independent interest given that all known multi-key FHE scheme (and all our MKHSS scheme from Section 4) have a quadratic blowup in the number of parties. Our alternative construction thus proves that this expansion is not inherent to "multi-keyness." However, we stress that our MKHSS construction still only works with two parties and still has a ciphertext size blowup relative to the non-multi-key HSS from DCR [OSY21]. Albeit this blowup comes from larger parameters and not from needing having more ciphertexts, in contrast to Section 4.3.

A.1 Overview of the alternative approach

In this section, we provide a brief overview of the main ideas behind our alternative approach to realizing multi-key HSS from DCR. In particular, we describe how we overcome the synchronization barriers described in Section 2 when using the Paillier–ElGamal encryption scheme.

First, in Appendix A.2 we explain how the parties can derive a common public key via the Diffie–Hellman protocol, and subtractive shares of the corresponding secret key. In contrast to the construction in Section 4, this approach synchronizes both the ciphertexts *and* the keys (instead of simply expanding the secret keys).

Second, in Appendix A.3, we explain how the two parties can locally use their derived keys to synchronize their respective input shares under their joint public key. This turns out to be more challenging, requiring some careful modifications to the original DCR-based HSS construction. However, in contrast to our constructions from Section 4, the parties no longer need to encrypt the "junk term," which reduces the number of ciphertexts in each input share.

A.2 Synchronizing keys

One trivial way for the parties to obtain a joint public key is to just perform a Diffie–Hellman key exchange. That is, the parties derive the common key $f := g^{-s_A \cdot s_B}$, given only each others' public keys $f_A := g^{-s_A}$ and $f_B := g^{-s_B}$. Here, g is a generator for a hidden-order subgroup of $\mathbb{Z}_{N^2}^*$ (e.g., the subgroup of quadratic residues). But how can the parties then obtain secret shares of the resulting secret key?

In retrospect, generating subtractive shares of the *joint* secret key (defined by multiplying the individual secret keys) is straightforward to achieve using existing tools. We show that we can use a minimal form of "non-interactive computation" [BM90, OSY21, CZ22], formalized as non-interactive multiplication (NIM) by Boyle et al. [BDSS25], which allows two parties to locally obtain subtractive secret shares of $s_A \cdot s_B$ using only their respective public keys. In particular, NIM can be seen as multi-key HSS for multiplication (or constant-degrees polynomials), which we then "bootstrap" into a multi-key HSS scheme supporting the computation of RMS programs.

Constructing NIM from DCR. At a high level, a NIM scheme allows Alice and Bob to generate shares of the multiplication of their respective inputs by exchanging one simultaneous message (or making these messages part of their respective MKHSS public keys, for example). A NIM scheme from the DCR assumption can be realized by following the blueprint laid out in several recent works [OSY21, ARS24, BCM+24] (we sketch the construction in Appendix A.4.1). Then, given a NIM scheme, Alice and Bob can locally derive subtractive shares of the joint secret key $s = s_A \cdot s_B$ (defined over the integers) using just the public key of the other party, which is exactly a share of the joint secret key they require.

A.3 Synchronizing input shares

We now explain how we overcome the challenges associated with synchronizing the HSS input shares. We focus on explaining how Alice and Bob can synchronize an HSS input generated by Alice under her public key pk_A (synchronizing Bob's input follows by reversing roles).

Using Paillier–ElGamal along with the "flipped" encryption trick we described in Section 2, an input share of a message x under Alice's public key is of the form:

$$\Big(((N+1)^x g^{r_A}, f_A^{r_A}),\ (g^{r'_A}, (N+1)^x f_A^{r'_A})\Big).$$

In particular, $(N+1)$ is the generator of the subgroup of $\mathbb{Z}^*_{N^2}$ in which computing the discrete logarithm is easy. This makes it possible to compute the DDLog efficiently base-$(N+1)$.

Recall that the goal of synchronization in MKHSS is to take an HSS input share of x generated under Alice's public key and transform it into an HSS input share under the joint key derived by Alice and Bob. In this case, Alice and Bob need to locally obtain an input share of the form:

$$\Big(((N+1)^x g^{r_A}, f^{r_A}),\ (g^{r'_A}, (N+1)^x f^{r'_A})\Big),$$

where $f := g^{-s}$ is the synchronized common public key and $s := s_A \cdot s_B$ is the joint secret key.

Towards synchronization. First, we note that Alice can trivially synchronize her own share by simply re-encrypting x under the joint public key f and reusing the same randomness r_A, r'_A she used to generate the original share. By doing so, Alice obtains a new HSS input share of the form:

$$[\![x]\!]_A := \Big(((N+1)^x \cdot g^{r_A}, f^{r_A}),\ (g^{r'_A}, (N+1)^x f^{r'_A})\Big), \tag{5}$$

which is distributed exactly as an input share under the joint public key f.

Now, we try and let Bob synchronize by computing

$$\Big(((N+1)^x \cdot g^{r_A}, (f_A^{r_A})^{s_B}),\ (g^{r'_A}, ((N+1)^x f_A^{r'_A})^{s_B})\Big)$$
$$= \Big(((N+1)^x \cdot g^{r_A}, f^{r_A}),\ (g^{r'_A}, (N+1)^{x\cdot s_B} f^{r'_A})\Big).$$

As already pointed out in Section 2, this "natural" approach to synchronization does not yield the desired encryption of x (the second component is an encryption of $x \cdot s_B$ and not an encryption of x). While we resolve this in our other constructions by expanding the secret keys and having Alice provide an additional encryption of the "junk term," we find a much simpler approach for synchronizing when using Paillier–ElGamal.

Specifically, our idea is to change the space from which the secret keys are sampled. Rather than sampling s_σ from the set $\{1, \ldots, N\}$, we instead sample the secret keys from the set

$$\{N+1,\ 2N+1,\ 3N+1,\ \ldots,\ (N-1)N+1\},$$

which guarantees that all sampled secret keys satisfy $s_\sigma \equiv 1 \bmod N$. Observe that because the secret keys are sampled over the integers, this requirement can be easily satisfied by first sampling $s'_\sigma \leftarrow\$ \{1, \ldots, N-1\}$ and then defining $s_\sigma := s'_\sigma \cdot N + 1 \in \mathbb{Z}$. While at first glance this may appear to be an odd choice for sampling the secret keys, it turns out to be just the trick for "automatically" canceling out the junk term created by Bob's attempt at synchronization (and does not harm security, as we will show later).

Specifically, using the fact that the secret key is congruent to 1 mod N and the fact that $(N+1)$ has order N in $\mathbb{Z}^*_{N^2}$, we get that $(N+1)^{s_B} = (N+1) \in \mathbb{Z}^*_{N^2}$. This property allows Bob to then synchronize the encryption of the message x as above because $((N+1)^x f_A^{r_A})^{s_B} = (N+1)^x f^{r_A}$, which is a proper encryption of x under public key f with randomness r_A, and matches Alice's synchronized encryption of x computed in Equation (5).

The high level intuition for why sampling the key in this way does not impact security is the following. First, observe that the public key g^{-s} in Paillier–ElGamal, computed with a secret key $s \leftarrow\$ [N]$, is close to a random subgroup element generated by g. Then, because g has order $\phi(N)/4$, a public key $g^{-s'}$ computed with $s' := N \cdot s$, is statistically close to g^{-s}, since $s \bmod \phi(N)$ is statistically close to $s \cdot N \bmod \phi(N)$. As such, the new sampling results in a public key that is *statistically* close to a standard Paillier–ElGamal public key.

Achieving full synchronization. We are now in a situation where, on the one hand, we need to sample the secret keys s_A and s_B such that $s_A \pmod N \equiv s_B \pmod N \equiv 1 \pmod N$ in order to allow Bob to locally synchronize the encryption of x. On the other hand, we wish to maintain the ability to compute multiplicative shares of $(N+1)^{x \cdot s}$, using the "flipped" decryption trick, without the secret key being canceled out by the order of the group.

Despite these two requirements appearing mutually exclusive, our next insight allows us to have our cake and eat it too. Instead of encrypting messages exclusively in $\mathbb{Z}^*_{N^2}$, we can encrypt them both in $\mathbb{Z}^*_{N^2}$ *and* in $\mathbb{Z}^*_{N^{w+1}}$, for some $w > 2$, by using the generalized Paillier–ElGamal encryption scheme of Damgård and Jurik [DJ03]. In $\mathbb{Z}^*_{N^{w+1}}$, the group element $(N+1)$ has order N^w, which allows us to encrypt x separately in $\mathbb{Z}^*_{N^2}$ and then duplicate this in $\mathbb{Z}^*_{N^{w+1}}$, such that $(N+1)^{x \cdot s} \in \mathbb{Z}^*_{N^2} \equiv (N+1)^x$ and $(N+1)^{x \cdot s} \in \mathbb{Z}^*_{N^{w+1}} \equiv (N+1)^{x \cdot s \pmod{N^w}}$, for sufficiently large w so that $x \cdot s$ does not exceed N^w. This allows us to satisfy both requirements. Additionally, we note that we are not limited to sampling a short secret key s, and so our construction does not necessitate making the short-exponent discrete logarithm assumption (in contrast to our constructions from the NIDLS framework in Section 4.3).

A.4 Alternative construction of MKHSS from DCR

In this section, we present the full MKHSS construction from DCR. Our construction uses two building blocks: NIM and the Damgård–Jurik encryption scheme, which we describe in Appendix A.4.1.

A.4.1 Building blocks Here, we describe the two building blocks that we use in our construction.

Damgård–Jurik–ElGamal encryption scheme. We first recall the Damgård–Jurik "ElGamal" encryption scheme in Figure 18. The scheme is proven secure under the DCR assumption [DJ03] (see also [CS02, BCP03]). For convenience, we extend the scheme to support the "flipped" encryptions via a FlipEncrypt algorithm.

For completeness, we prove the security of the extended DJEG encryption scheme presented in Figure 18.

Assumption 1 (Decisional Composite Residuosity (DCR) Assumption)**.** *Let* GenPQ *be a randomized algorithm that, on input the security parameter λ, outputs two distinct, sufficiently large, random safe primes p and q. The DCR assumption states that:*

$$\left\{ (N, g_0) \;\middle|\; \begin{array}{r} (p,q) \leftarrow \mathsf{GenPQ}(1^\lambda) \\ N := pq \\ g_0 \leftarrow\!\$\; \mathbb{Z}^*_{N^2} \end{array} \right\} \approx_c \left\{ (N, g_1) \;\middle|\; \begin{array}{r} (p,q) \leftarrow \mathsf{GenPQ}(1^\lambda) \\ N := pq \\ g_0 \leftarrow\!\$\; \mathbb{Z}^*_{N^2} \\ g_1 := g_0^N \end{array} \right\}.$$

Lemma 5. *Let λ be a security parameter. If the DCR assumption holds, then the encryption scheme presented in Figure 18 satisfies the standard notion of semantic security (i.e., CPA-security).*

Proof. The proof of semantic security follows a similar proof made in [BCCS24, Section 4.4] and proceeds with a simple hybrid argument. Here, we adapt the proof to the generalized Damgård–Jurik–ElGamal setting.

– *Hybrid $\mathcal{H}_0$.* This hybrid consist of a ciphertext (c_0, c_1) as generated by DJEG.Encrypt in Figure 18.

– *Hybrid $\mathcal{H}_1$.* In this hybrid, we change how the randomness r is sampled in DJEG.Encrypt, and sample r uniformly from $\{0, 1, \ldots, N^{w+1}\}$ instead of $\{0, 1, \ldots, N\}$.

  Claim. $\mathcal{H}_1 \approx_s \mathcal{H}_0$.

  Proof. This hybrid is statistically close to the previous one by the fact that g and f have order $\phi(N)/4$, which is coprime to N. We note that we implicitly use the fact that GenPQ outputs safe primes making g, as sampled in Figure 18, a generator for the subgroup of order $\phi(N)/4$ with overwhelming probability. □

- *Hybrid $\mathcal{H}_2$.* In this hybrid, we change how the public key f is sampled in $\mathsf{DJEG.KeyGen}$ by sampling f as a uniformly random $2N$-th residue. That is, $f := (g')^{2N} \in \mathbb{Z}^*_{N^2}$, where $g' \leftarrow\$ \mathbb{Z}^*_{N^2}$.

  Claim. $\mathcal{H}_2 \approx_s \mathcal{H}_1$.

  Proof. By the definition of $g = (g_0)^{2N}$, it is a generator for the subgroup of the $2N$-th residues with overwhelming probability (again, using the fact that N is a composite of safe primes). Then, it suffices to note that in $\mathcal{H}_1$ we have $f = g^s$, which is a uniformly random $2N$-th residue when g is a generator for the subgroup of $2N$-th residues and s is sampled uniformly from $\mathbb{Z}_N$. □

- *Hybrid $\mathcal{H}_3$.* In this hybrid, we change how the public key f is sampled in $\mathsf{DJEG.KeyGen}$ by sampling f as a uniformly random square from $\mathbb{Z}^*_{N^2}$.

  Claim. $\mathcal{H}_3 \approx_c \mathcal{H}_2$ assuming DCR.

  Proof. The claim follows from a direct reduction to the DCR assumption. Notice that f is sampled as a $2N$-th residue in $\mathcal{H}_2$ and a random square of $\mathbb{Z}^*_{N^2}$ in $\mathcal{H}_3$. The reduction thus has at most a factor of two loss in advantage in the DCR game. □

  Remark 9. We note that, thanks to CRT decomposition, $\mathbb{Z}_{N^w \cdot \phi(N)}$ is isomorphic to $\mathbb{Z}_{N^w} \times Z_{\phi(N)}$ because N is coprime to $\phi(N)$. Using this, any element c in $\mathbb{Z}^*_{N^{w+1}}$ can be written as $c = (1+N)^a g^b \bmod N^{w+1}$, for some $(a,b) \in \mathbb{Z}_{N^w} \times \mathbb{Z}_{\phi(N)/4}$, since all elements in $\mathbb{Z}^*_{N^w \cdot \phi(N)}$ can be decomposed into this form. Moreover, for a random c, with overwhelming probability $1 - \frac{p+q-1}{N}$, we have that $a \neq 0$ and coprime to N.

- *Hybrid $\mathcal{H}_4$.* In this hybrid, the ciphertext elements c_0 and c_1 are sampled as uniformly random elements of $\mathbb{Z}^*_{N^{w+1}}$.

  Claim. $\mathcal{H}_4 \approx_s \mathcal{H}_3$.

  Proof. We claim that, in hybrid $\mathcal{H}_3$, (c_0, c_1) is already statistically close to the uniform distribution over $\mathbb{Z}^*_{N^{w+1}} \times \mathbb{Z}^*_{N^{w+1}}$. To see this, we first note that g generates a subgroup of order $\phi(N)/4$ and, therefore, the element c_0 *statistically* reveals only the value $r_0 = r \bmod \phi(N)/4$. Moreover, using Remark 9, c_1 can be rewritten as follows:

  $$c_1 = (1+N)^x \cdot f^r = (1+N)^{ar_1 + x \bmod N^w} \cdot g^{b \cdot r_0 \bmod \phi(N)/4} \bmod N^{w+1},$$

  where $r_0 = r \bmod \phi(N)/4$ and $r_1 = r \bmod N^w$. Then, conditioned on r_0, r_1 is statistically close to a uniformly random element by the fact that N and $\phi(N)$ are coprime. By the above, we have that $ar_1 + x \bmod N^w$ is statistically close to a uniformly random element of $\mathbb{Z}_{N^w}$ given r_0 (recall that a is coprime to N, with overwhelming probability). Combined, we have that (c_0, c_1) are statistically close to a uniformly random tuple of elements sampled from $\mathbb{Z}^*_{N^{w+1}}$. □

We have now concluded the proof of semantic security for the DJEG scheme when the ciphertext is generated using $\mathsf{DJEG.Encrypt}$. We note that a very similar hybrid argument applies to proving that ciphertexts output by the "flipped" encryption $\mathsf{DJEG.FlipEncrypt}$ are computationally indistinguishable from uniform under the DCR assumption. A little more formally, starting with $\mathcal{H}_3$, the element f is distributed identically to g (both are random squares in $\mathbb{Z}^*_{N^2}$) which enables interchanging them in c_0 and c_1. This concludes the proof. ■

Non-interactive multiplication. Here, we sketch non-interactive multiplication (NIM), as defined by Boyle, Devadas, and Servan-Schreiber [BDSS25]. We define the NIM syntax to be role-agnostic (following Remark 4), which simplifies the presentation in the MKHSS construction.

Definition 16 (Non-Interactive Multiplication; Adapted from [BDSS25])**.** *Let λ be a security parameter, $\mathcal{R}$ be a finite ring. A non-interactive multiplication (NIM) scheme consists of three algorithms* $\mathsf{NIM} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *with the following syntax:*

Damgård–Jurik–ElGamal Encryption Scheme [DJ03]

Public Parameters. A generator GenPQ with respect to which the DCR assumption holds.

```
$\mathsf{DJEG.Setup}(1^\lambda)$:
  1 : $(p, q) \leftarrow \mathsf{GenPQ}(1^\lambda)$
  2 : $N := pq$
  3 : $g_0 \leftarrow\$ \mathbb{Z}^*_{N^2}$
  4 : $g := (g_0)^{2N} \in \mathbb{Z}^*_{N^2}$
  5 : return $\mathsf{crs} := (N, g)$

$\mathsf{DJEG.KeyGen}(\mathsf{crs})$:
  1 : parse $\mathsf{crs} = (N, g)$
  2 : $s \leftarrow\$ [N]$
  3 : $f := g^s$
  4 : $(\mathsf{pk}, \mathsf{sk}) := (f, s)$
  5 : return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{DJEG.Encrypt}(\mathsf{crs}, \mathsf{pk}, x, w)$:
  1 : parse $\mathsf{crs} = (N, g)$
  2 : parse $\mathsf{pk} = f$
  3 : $r \leftarrow\$ \{0, 1, \dots, N\}$
  4 : $c_0 := g^r \bmod N^{w+1}$
  5 : $c_1 := (N + 1)^x f^r \bmod N^{w+1}$
  6 : return $\mathsf{ct} := (c_0, c_1)$

$\mathsf{DJEG.FlipEncrypt}(\mathsf{crs}, \mathsf{pk}, x, w)$:
  1 : parse $\mathsf{crs} = (N, g)$
  2 : parse $\mathsf{pk} = f$
  3 : $r \leftarrow\$ \{0, 1, \dots, N\}$
  4 : $c_0 := (N + 1)^x g^r \bmod N^{w+1}$
  5 : $c_1 := f^r \bmod N^{w+1}$
  6 : return $\mathsf{ct} := (c_0, c_1)$

$\mathsf{DJEG.Decrypt}(\mathsf{sk}, \mathsf{ct})$:
  1 : parse $\mathsf{ct} = (c_0, c_1)$
  2 : $c' := c_1/(c_0)^{\mathsf{sk}}$
  3 : $x := \frac{c' - 1}{N^w}$
  4 : return $x$
```

Fig. 18: The DJEG encryption scheme.

– $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string* crs.
– $\mathsf{Encode}(\mathsf{crs}, x) \to (\mathsf{pe}_\sigma, \mathsf{st}_\sigma)$. *The randomized encoding algorithm takes as input the CRS* crs *and a ring element* $x \in \mathcal{R}$. *It outputs a public encoding* pe_σ *and secret state* st_σ.
– $\mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_{1-\sigma}, \mathsf{st}_\sigma) \to \langle z \rangle_\sigma$. *The deterministic decoding algorithm takes as input the CRS* crs, *another party's public encoding* $\mathsf{pe}_{1-\sigma}$, *and secret state* st_σ. *It outputs a subtractive secret share of* z.

The above functionality must satisfy correctness and security, which are defined as follows:

Correctness. For all security parameters $\lambda \in \mathbb{N}$ *and every pair of elements* $x, y \in \mathcal{R}$, *a NIM scheme is said to be correct if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\langle z \rangle_A - \langle z \rangle_B = xy \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}(\mathsf{crs}, x) \\ (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}(\mathsf{crs}, y) \\ \langle z \rangle_A := \mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A) \\ \langle z \rangle_B := \mathsf{Decode}(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. A NIM scheme is said to be secure *if for all efficient adversaries* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, *and all* $\sigma \in \{A, B\}$, *we have that*

$$\Pr\left[b' = b \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow\$ \{0, 1\} \\ (\mathsf{pe}_\sigma, \mathsf{st}_\sigma) \leftarrow \mathsf{Encode}(\mathsf{crs}, x_b) \\ b' \leftarrow \mathcal{A}(\mathsf{pe}_\sigma, \mathsf{st}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $x_0, x_1 \in \mathcal{R}$.

Sketch: Constructing NIM from DCR. Here, we briefly sketch the construction of NIM from the DCR assumption from Boyle et al. The scheme is essentially a simplification of non-interactive VOLE [OSY21] from the DCR assumption. In a nutshell, the idea is to first compute the multiplication "in the exponent" of the group and then compute the DDLog to obtain subtractive shares over the integers.

NIM from DCR. Let N be a suitable composite modulus and let g and h be random generators of $\mathbb{Z}^*_{N^2}$ that are part of the CRS. The protocol is instantiated over the ring $\mathcal{R} = \mathbb{Z}_\ell$, where for correctness we need $\ell < 2^{-\lambda} \cdot \sqrt{N}$. The high level idea behind the NIM construction is to have:

– Alice's public encoding consist of a Pedersen-like commitment $g^{r_A} h^x$ to her element x and
– Bob's public encoding consist of an encryption $(g^{r_B}, (N+1)^y h^{r_B})$ of his element y,

where r_A and r_B are random elements of $\mathbb{Z}_N$.

Then, given Alice's encoding $\mathsf{pe}_A := g^{r_A} h^x$, Bob derives $Z_B := (g^{r_A} h^x)^{-r_B} = g^{-r_A r_B} h^{-x r_B}$. Similarly, given Bob's encoding $\mathsf{pe}_B := (g^{r_B}, (N+1)^y h^{r_B})$, Alice derives $Z_A := (g^{r_B})^{r_A} \cdot ((N+1)^y h^{r_B})^x$. It's not hard to see that Z_A and Z_B form multiplicative shares of $(N+1)^{xy \bmod N}$ since:

$$\begin{aligned} Z_A \cdot Z_B &= ((g^{r_B})^{r_A} \cdot ((N+1)^y h^{r_B})^x) \;\cdot\; (g^{r_A} h^x)^{-r_B} \\ &= (g^{r_A r_B} \cdot (N+1)^{xy} h^{x r_B}) \;\cdot\; (g^{-r_A r_B} h^{-x r_B}) \\ &= (N+1)^{xy}. \end{aligned}$$

Therefore, by applying the DDLog procedure to Z_A and Z_B, the parties recover subtractive shares of xy mod N. Moreover, because $x, y < 2^{-\lambda} \cdot \sqrt{N}$, we have that, with all but negligible probability, the shares $\langle xy \rangle_A$ and $\langle xy \rangle_B$ are subtractive shares over the integers, by the correctness of the DDLog algorithm.

A.4.2 Alternative MKHSS construction

We present the alternative MKHSS construction in Figure 19. Each party samples a secret key $s_\sigma \leftarrow\$ \{i \cdot N + 1 \mid 1 \le i \le N-1\}$ such that $s_\sigma \equiv 1 \bmod N$. The public key pk_σ of each party consists of the group element $f_\sigma := g^{-s_\sigma}$ and public NIM encoding of s_σ. Alice and Bob then synchronize their keys and respective input shares as described in the overview. In particular, because the input shares are nearly identical to the input shares in the Paillier–ElGamal constructions of (non-multi-key) HSS [OSY21, RS21], the correctness of evaluation for computing RMS programs is almost immediate. Moreover, security reduces to the semantic security of the Damgård–Jurik encryption scheme and the NIM scheme.

Concrete performance estimates. We note that the construction in Figure 19 is potentially implementable. Finding ways to further optimize it is an interesting direction for future work. As an immediate optimization, we make the short exponent assumption and sample the keys from a shorter space, allowing us to work in the group $\mathbb{Z}^*_{N^4}$ instead of $\mathbb{Z}^*_{N^6}$. Then, the main overhead of Figure 19 is exponentiation in $\mathbb{Z}^*_{N^4}$. Each RMS multiplication in our construction requires two exponentiations: one exponentiation in $\mathbb{Z}^*_{N^4}$ and one in $\mathbb{Z}^*_{N^2}$, which require roughly 60 milliseconds and 15 milliseconds on high-end hardware, respectively, when using a 3072-bit modulus N. Therefore, we can expect each multiplication to take between 75 and 100 milliseconds. This results in roughly 10 multiplications per second. In contrast, (non-multi-key) HSS can achieve upwards of 100 multiplications per second on high-end hardware [BCG+17], making our construction an order of magnitude slower.

Theorem 6. *Let λ is the security parameter and let $N = N(\lambda)$ be the output of* GGen *as defined in Figure 18. If the DCR assumption holds, then the construction described in Figure 19 is an MKHSS scheme for the class of polynomial sized RMS programs with bound $B < 2^{-\lambda} \cdot N$ and message space $\mathbb{Z}_B$.*

Alternative Construction of MKHSS from DCR

Public Parameters. Let $S_{\mathsf{sk}} := \{i \cdot N + 1 \mid 1 \le i \le N-1\}$ be the secret key space and let B be a bound on the message space. Let $\mathsf{NIM} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ be a NIM scheme. We will use the algorithms $\mathsf{ExpLinEncS}$ and $\mathsf{ExpLinEncR}$ defined in Figure 20.

```
$\mathsf{MKHSS.Setup}(1^\lambda, w)$:
  1 : $(N, g) \leftarrow \mathsf{DJEG.Setup}(1^\lambda)$
  2 : $\mathsf{crs}_{\mathsf{nim}} \leftarrow \mathsf{NIM.Setup}(1^\lambda)$
  3 : $k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}} \leftarrow\$ \{0,1\}^\lambda$
  4 : $\mathsf{crs} := (N, g, \mathsf{crs}_{\mathsf{nim}}, k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}})$
  5 : **return** $\mathsf{crs}$
```

```
$\mathsf{MKHSS.KeyGen}(\mathsf{crs})$:
  1 : **parse** $(N, g, \mathsf{crs}_{\mathsf{nim}})$ **from** $\mathsf{crs}$
  2 : $s \leftarrow\$ S_{\mathsf{sk}},\ f := g^{-s}$
  3 : $(\mathsf{pe}, \mathsf{st}) \leftarrow \mathsf{NIM.Encode}(\mathsf{crs}_{\mathsf{nim}}, s)$
  4 : $\mathsf{pk} := (\mathsf{pe}, f)$
  5 : $\mathsf{sk} := (\mathsf{st}, s)$
  6 : **return** $(\mathsf{pk}, \mathsf{sk})$
```

```
$\mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$:
  1 : **parse** $\mathsf{crs} = (N, g, \mathsf{crs}_{\mathsf{nim}})$
  2 : **parse** $\mathsf{pk}_\sigma = (\mathsf{pe}_\sigma, f_\sigma)$
  3 : $r, r' \leftarrow\$ \mathbb{Z}_N$
  4 : $\mathsf{ct} \leftarrow \mathsf{DJEG.FlipEncrypt}(f_\sigma, x, 5; r)$
  5 : $\mathsf{ct}' \leftarrow \mathsf{DJEG.Encrypt}(f_\sigma, x, 1; r')$
  6 : $[\![x]\!]_\sigma^\sigma := \big((x, r, r'), (\mathsf{ct}, \mathsf{ct}')\big)$
  7 : $[\![x]\!]_{1-\sigma}^\sigma := (\mathsf{ct}, \mathsf{ct}')$
  8 : **return** $([\![x]\!]_A^\sigma, [\![x]\!]_B^\sigma)$
```

```
$\mathsf{MKHSS.Eval}(\mathsf{crs}, \sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![\mathbf{x}_A]\!]_\sigma^A, [\![\mathbf{x}_B]\!]_\sigma^B, P)$:
  1 : **parse** $(\mathsf{crs}_{\mathsf{nim}}, k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}})$ **from** $\mathsf{crs}$
  2 : **parse** $\mathsf{sk}_\sigma = (\mathsf{st}_\sigma, s_\sigma)$
  3 : **parse** $\mathsf{pk}_{1-\sigma} = (\mathsf{pe}_{1-\sigma}, f_{1-\sigma})$
  4 : $f := (f_{1-\sigma})^{s_\sigma}$
  5 : $\langle z \rangle_\sigma := \mathsf{NIM.Decode}(\mathsf{crs}_{\mathsf{nim}}, \mathsf{pe}_{1-\sigma}, \mathsf{st}_\sigma)$
  6 : $\mathbf{k}_\sigma := (\langle z \rangle_\sigma, 1)$ **if** $\sigma = A$ **else** $\mathbf{k}_\sigma := (\langle z \rangle_\sigma, 0)$
  7 : **for** $i \in [m]$:
  8 :     $\{\!\{x_\sigma^{(i)}\}\!\} := \mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x_\sigma^{(i)}]\!]_\sigma^\sigma)$
  9 :     $\{\!\{x_{1-\sigma}^{(i)}\}\!\} := \mathsf{ExpLinEncR}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x_{1-\sigma}^{(i)}]\!]_\sigma^{1-\sigma})$
 10 : $\mathsf{ek}_\sigma := (k_1^{\mathsf{prf}}, k_2^{\mathsf{prf}}, \mathbf{k}_\sigma)$
 11 : $\{\!\{\mathbf{x}\}\!\} := (\{\!\{x_A^{(1)}\}\!\}, \ldots, \{\!\{x_A^{(m)}\}\!\}, \{\!\{x_B^{(1)}\}\!\}, \ldots, \{\!\{x_B^{(m)}\}\!\})$
 12 : **return** $\mathsf{DEval}(\sigma, \mathsf{ek}_\sigma, \{\!\{\mathbf{x}\}\!\}, P)$
```

Fig. 19: Alternative Construction of MKHSS from DCR.

```
$\mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]_\sigma^\sigma)$:
  1 : **parse** $[\![x]\!]_\sigma^\sigma = \big((x, r, r'), (\_, \_)\big)$
  2 : **parse** $\mathsf{sk}_\sigma := (\_, s_\sigma)$
  3 : **parse** $\mathsf{pk}_{1-\sigma} = (\_, f_{1-\sigma})$
  4 : $(c_0, c_1) := \mathsf{DJEG.FlipEncrypt}(f_{1-\sigma}, x, w; r)$
  5 : $(c_0', c_1') := \mathsf{DJEG.Encrypt}(f_{1-\sigma}, x, 2; r')$
  6 : $\{\!\{x\}\!\} := ((c_0, (c_1)^{s_\sigma}), (c_0', (c_1')^{s_\sigma}))$
  7 : **return** $\{\!\{x\}\!\}$
```

```
$\mathsf{ExpLinEncR}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]_\sigma^{1-\sigma})$:
  1 : **parse** $[\![x]\!]_\sigma^{1-\sigma} = \big((c_0, c_1), (c_0', c_1')\big)$
  2 : **parse** $\mathsf{sk}_\sigma = (\_, s_\sigma)$
  3 : $\{\!\{x\}\!\} := ((c_0, (c_1)^{s_\sigma}), (c_0', (c_1')^{s_\sigma}))$
  4 : **return** $\{\!\{x\}\!\}$
```

Fig. 20: Exponent-linear encoding algorithms used as subroutines in the DCR-based MKHSS construction.

Proof. We prove correctness and privacy in turn.

Correctness. Recall that the correctness property requires that parties obtain a subtractive sharing of the program output upon evaluation.

We first prove that the encoding $\{\!\{x\}\!\}$ derived by the parties in $\mathsf{MKHSS.Eval}$ is (1) the same for both parties and (2) exponent-linear decodable.

Claim. For all integers $x \in \mathbb{Z}_N$ and all $\sigma \in \{A, B\}$, we have

$$\{\!\{x\}\!\} = \mathsf{ExpLinEncS}(\mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma}, [\![x]\!]^\sigma_\sigma) = \mathsf{ExpLinEncR}(\mathsf{sk}_{1-\sigma}, \mathsf{pk}_\sigma, [\![x]\!]^\sigma_{1-\sigma}),$$

where $([\![x]\!]^\sigma_A, [\![x]\!]^\sigma_B) \leftarrow \mathsf{MKHSS.Share}(\mathsf{crs}, \sigma, \mathsf{pk}_\sigma, x)$. Moreover, $\{\!\{x\}\!\}$ is base-$(N+1)$ exponent-linear decodable under the decoding key $\mathbf{k} = (s_A \cdot s_B,\ 1)$.

Proof. We consider the case where $\sigma = A$; a symmetric argument follows for the case where $\sigma = B$.

By inspecting $\mathsf{MKHSS.Share}$, we have $[\![x]\!]^A_A = ((x, r, r'), (\mathsf{ct}, \mathsf{ct}'))$ and $[\![x]\!]^A_B = (\mathsf{ct}, \mathsf{ct}')$, where

$$\mathsf{ct} = ((N+1)^x \cdot g^r,\ f_A^r) \text{ and } \mathsf{ct}' = (g^{r'},\ (N+1)^x \cdot f_A^{r'}).$$

Party-A computes $\{\!\{x\}\!\}$ in $\mathsf{ExpLinEncS}$ as

$$\begin{aligned}\{\!\{x\}\!\} &= \Big(((N+1)^x \cdot g^r,\ (f_B^r)^{s_A}),\ (g^{r'},\ ((N+1)^x \cdot f_B^{r'})^{s_A})\Big) \in (\mathbb{Z}^*_{N^6})^2 \times (\mathbb{Z}^*_{N^2})^2 \\ &= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^{x \cdot s_A} \cdot f_B^{r' \cdot s_A})\Big) \\ &= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^x \cdot f^{r'})\Big),\end{aligned}$$

where the third equality follows from the fact that $s_A \equiv 1 \pmod N$ and $(N+1)$ has order N in $\mathbb{Z}^*_{N^2}$. Now, observe that party-B computes $\{\!\{x\}\!\}$ in $\mathsf{ExpLinEncR}$ as

$$\begin{aligned}\{\!\{x\}\!\} &= \Big(((N+1)^x \cdot g^r,\ (f_A^r)^{s_B}),\ (g^{r'},\ ((N+1)^x \cdot f_A^{r'})^{s_B})\Big) \in (\mathbb{Z}^*_{N^6})^2 \times (\mathbb{Z}^*_{N^2})^2 \\ &= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^{x \cdot s_B} \cdot f_A^{r' \cdot s_B})\Big) \\ &= \Big(((N+1)^x \cdot g^r,\ f^r),\ (g^{r'},\ (N+1)^x \cdot f^{r'})\Big).\end{aligned}$$

Therefore, both parties obtain the same encoding $\{\!\{x\}\!\}$.

We are left to show that $\{\!\{x\}\!\} = (\mathbf{c}_0, \mathbf{c}_1)$ is base-$(N+1)$ exponent-linear decodable under $\mathbf{k} = (k_1, k_2) = (s_A \cdot s_B,\ 1)$. Observe that

$$\begin{aligned}\langle \mathbf{c}_0, \mathbf{k}\rangle &= ((N+1)^x \cdot g^r)^{s_A \cdot s_B} \cdot f^r = (N+1)^{x \cdot s_A \cdot s_B} \cdot g^{r \cdot s_A \cdot s_B} \cdot g^{-s_A \cdot s_B \cdot r} = (N+1)^{x \cdot s_A \cdot s_B} \in \mathbb{Z}^*_{N^6} \\ \langle \mathbf{c}_1, \mathbf{k}\rangle &= ((g^{r'})^{s_A \cdot s_B} \cdot (N+1)^x \cdot f^{r'} = g^{r \cdot s_A \cdot s_B} \cdot (N+1)^x \cdot g^{-s_A \cdot s_B \cdot r} = (N+1)^x \in \mathbb{Z}^*_{N^2},\end{aligned}$$

which proves that $\{\!\{x\}\!\}$ is base-$(N+1)$ exponent-linear decodable. In particular, $s_A \cdot s_B \le ((N-1) \cdot N)^2 < N^4$. Furthermore, because $x \le N$, we have that $x \cdot s_A \cdot s_B$ does not overflow modulo N^5. □

Finally, to complete the proof of correctness, it suffices to note that by the correctness of NIM, the parties obtain subtractive shares of $s_A \cdot s_B$, and so $\mathbf{k}_\sigma$ is a subtractive share of $\mathbf{k}$ as defined above.

In sum, it follows that parties run DEval with encodings of the input that are base-$(N+1)$ exponent-linear decodable. Finally, since $B < 2^{-\lambda} \cdot N$ and DDLog is a B-bounded (resp. $(B \cdot N^4)$-bounded) base-$(N+1)$ algorithm for distributed discrete logarithm with negligible correctness error in $\mathbb{Z}^*_{N^2}$ (resp. $\mathbb{Z}^*_{N^6}$), it follows from Lemma 3 that the MKHSS scheme satisfies the correctness property for all polynomial-size RMS programs P.

Security. Recall that the security property requires that the input share $[\![x]\!]^\sigma_{1-\sigma}$ of party-$(1-\sigma)$, ensures the privacy of an input x shared using party-σ's public key pk_σ.

Consider any efficient adversary $\mathcal{A}$ for the security experiment defined in Definition 6. Let the output of the security experiment be defined as 1 if $\mathcal{A}$'s output b' is equal to the challenge bit b; else let the output of the experiment be defined as 0. We will use a hybrid argument to show that the output of the experiment is 1 with probability of at most $1/2 + \mathsf{negl}(\lambda)$.

– *Hybrid* $\mathcal{H}_0$. This hybrid consists of the output of the experiment when run with adversary $\mathcal{A}$ when the challenge bit is $b = 0$.

– *Hybrid* $\mathcal{H}_1$. This hybrid game is identical to the previous hybrid, except that the secret key s_σ is sampled uniformly at random from $[N]$ in $\mathsf{MKHSS.KeyGen}$, which matches the distribution of the secret key in the DJEG encryption scheme.

  Claim. $\mathcal{H}_1 \approx_s \mathcal{H}_0$.

  Proof. Note that in $\mathcal{H}_0$ the public key is computed as $f = g^{N \cdot i} g$ for some $i \in [N-1]$ while in $\mathcal{H}_1$ it is computed as g^i for $i \in [N]$. Because g has order $\phi(N)/4$, and for a random $i \in [N-1]$, $i \pmod{\phi})(N)/4$ is statistically close to $i \cdot N \bmod \phi(N)/4$ (since $\phi(N)$ is co-prime to N), it follows that g^i and $(g^N)^i$ are both statistically close to the uniform distribution. To conclude the proof, it suffices to note that $(g^N)^i \cdot g$ is also close to uniform. □

– *Hybrid* $\mathcal{H}_2$. In this hybrid game, pe is replaced with an encoding of zero. That is, $(\mathsf{pe}_\sigma, \_) \leftarrow \mathsf{NIM.Encode}(\mathsf{crs}_{\mathsf{nim}}, 0)$.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ assuming the security of NIM.

  Proof. The claim follows immediately from the security of the NIM scheme. □

– *Hybrid* $\mathcal{H}_3$. In this hybrid game, we replace the DJEG encryptions with encryptions of x_1.

  Claim. $\mathcal{H}_3 \overset{s}{\approx} \mathcal{H}_2$ by the semantic security of the DJEG encryption scheme.

  Proof. The claim follows by a straightforward hybrid argument replacing the two encryptions of x_0 with encryptions of x_1 and invoking the semantic security of the DJEG scheme. □

– *Hybrid* $\mathcal{H}_4$. In this hybrid game, we reverse the changes made in $\mathcal{H}_2$ and encode the secret key s using the NIM scheme.

  Claim. $\mathcal{H}_4 \approx_c \mathcal{H}_3$ assuming the security of NIM.

  Proof. The claim follows immediately from the security of the NIM scheme. □

– *Hybrid* $\mathcal{H}_5$. In this hybrid game, we reverse the changes made in $\mathcal{H}_1$ and sample the secret key as in the construction.

  Claim. $\mathcal{H}_5 \approx_c \mathcal{H}_4$.

  Proof. The proof follows the same argument used to prove that $\mathcal{H}_1 \approx_c \mathcal{H}_0$. □

To complete the proof, observe that $\mathcal{H}_5$ is exactly the output of the experiment when the challenge bit $b = 1$. Since we've shown that $\mathcal{H}_0 \approx_c \mathcal{H}_5$, it follows that $\mathcal{A}$ wins the MKHSS security game with probability of at most $1/2 + \mathsf{negl}(\lambda)$. ■