Succinct Homomorphic Secret Sharing*

Damiano Abram[0009−0004−3916−7550], Lawrence Roy, and Peter Scholl[0000−0002−7937−8422]

Aarhus University

damiano.abram@cs.au.dk
ldr709@gmail.com
peter.scholl@cs.au.dk

Abstract. This work introduces homomorphic secret sharing (HSS) with succinct share size. In HSS, private inputs are shared between parties, who can then homomorphically evaluate a function on their shares, obtaining a share of the function output. In succinct HSS, a portion of the inputs can be distributed using shares whose size is sublinear in the number of such inputs. The parties can then locally evaluate a function f on the shares, with the restriction that f must be linear in the succinctly shared inputs.

We construct succinct, two-party HSS for branching programs, based on either the decisional composite residuosity assumption, a DDH-like assumption in class groups, or learning with errors with a super-polynomial modulus-to-noise ratio. We then give several applications of succinct HSS, which were only previously known using fully homomorphic encryption, or stronger tools:

- **Succinct vector oblivious linear evaluation (VOLE):** Two parties can obtain secret shares of a long, arbitrary vector $\boldsymbol{x}$, multiplied by a scalar Δ, with communication sublinear in the size of the vector.
- **Batch, multi-party distributed point functions**: A protocol for distributing a batch of secret, random point functions among N parties, for any polynomial N, with communication sublinear in the number of DPFs.
- **Sublinear MPC for any number of parties:** Two new constructions of MPC with sublinear communication complexity, with N parties for any polynomial N: (1) For general layered Boolean circuits of size s, with communication $O(Ns/\log\log s)$, and (2) For layered, sufficiently wide Boolean circuits, with communication $O(Ns/\log s)$.

1 Introduction

Homomorphic secret sharing (HSS) allows two or more parties to perform a distributed computation on private inputs. HSS can be seen as a distributed analogue of homomorphic encryption, where instead of having a single server carry out a computation on encrypted inputs, several parties are each given a share of the inputs, and can then locally carry out homomorphic computations on the shares, to obtain a share of the desired result.

HSS was first introduced by Boyle, Gilboa and Ishai [BGI16], who showed how to build two-party HSS for branching programs based on the decisional Diffie-Hellman (DDH) assumption. Concretely, [BGI16] actually build HSS for a class of *restricted multiplication straightline* programs, or RMS programs, which are circuits with the restriction that every multiplication gate is between an input wire to the computation and one other intermediate wire. It was shown in [BGI16] that RMS programs are powerful enough to capture all logspace computations, which includes polynomial-sized branching programs.

A major application of the [BGI16] result was to obtain secure two-party computation for arbitrary, layered Boolean circuits of size s with a communication complexity of $O(s/\log s)$ bits. This was the first work

* Supported by the Aarhus University Research Foundation (AUFF), the Independent Research Fund Denmark (DFF) (grant DFF-0165-00107B "C3PO"), and the DARPA SIEVE program (contract HR001120C0085 "FROMAGER"). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

to bypass the circuit-size barrier in secure computation, without relying on fully homomorphic encryption (FHE). Since then, many subsequent works on HSS have focused on improving efficiency and obtaining HSS under new assumptions, offering alternative approaches to sublinear secure computation without FHE. For example, [OSY21] obtained an HSS construction based on Paillier encryption, which removed an inverse polynomial correctness error present in the original DDH-based construction, while [RS21] extended this to the Damgård-Jurik cryptosystem. Many works have shown other applications of HSS and its techniques, including building trapdoor hash functions [DGI+19,RS21], correlated pseudorandomness [BCG+19,OSY21] and more [BGI+18,BDG+22].

1.1 Our Results

In this work, we introduce a new family of *succinct HSS* schemes. In succinct HSS, a subset of the inputs from one party can be shared succinctly, with a share size of $o(n)$ for n-bit inputs. This is in contrast to classical HSS, where all known constructions have share size $\Omega(n)$. We build two-party, succinct HSS for the class of *special, restricted multiplication straightline* (special RMS) programs. A function f is a special RMS program if it is an RMS program, and furthermore if $f(x, y)$ is linear in x, where x is the subset of inputs with succinct shares.

Below is an overview of our main results, including applications to distributed point functions and MPC with sublinear communication.

Bilinear HSS and succinct VOLE. We start by building a form of HSS for bilinear functions, that is, two-party HSS where one party, Alice, holds as input a vector $\boldsymbol{x} \in \mathbb{Z}_q^n$, while Bob holds a matrix $M \in \mathbb{Z}_q^{m \times n}$. The goal is to obtain additive secret shares of the bilinear function $M \cdot \boldsymbol{x}$, using only a single round of interaction. We construct a succinct form of bilinear HSS, where Alice's message has size *independent of n and m*, using techniques from previous trapdoor hash functions for linear predicates [DGI+19, RS21]. We obtain instantiations based on each of the following assumptions: (1) decisional composite residuosity (DCR), (2) quadratic residuosity (QR), (3) a DDH-like assumption in class groups, or (4) learning with errors (LWE) with a superpolynomial modulus-to-noise ratio.

We show that bilinear HSS can be used to build a succinct form of *vector oblivious linear evaluation* (VOLE), that is, the functionality $f((\boldsymbol{x}_0, \boldsymbol{x}_1), \Delta) = \boldsymbol{x}_0\Delta + \boldsymbol{x}_1$, with input vectors $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ from Alice and a scalar Δ from Bob: we obtain a one-round protocol, where Alice's input $\boldsymbol{x}_1$ is sampled at random and the total communication is is sublinear in the dimension of $\boldsymbol{x}_0$. We call this primitive *succinct, half-chosen VOLE*. Half-chosen VOLE suffices for many applications of VOLE, such as designated-verifier zero-knowledge proofs [BMRS21, DIO21, YSWW21] and private set intersection [RR22].

Theorem 1 (informal). *There exists a protocol for half-chosen VOLE of length n, with one parallel message from each party, and total communication complexity $O(n^{2/3})$,[1] if one of the following assumptions holds: (1) DCR, (2) QR, (3) a DDH-like assumption in class groups, or (4) LWE with a superpolynomial modulus-to-noise ratio.*

The communication can further be reduced to $O(n^{1/2})$, by relying on a variant of bilinear HSS for structured matrices with reduced communication cost. This requires either the power-DDH assumption over Paillier groups or class groups, or a new variant of ring-LWE called the *power ring-LWE* assumption.

We note that previous constructions of laconic function evaluation (LFE) also imply succinct, bilinear HSS and half-chosen VOLE. However, LFE constructions are only known from LWE with a sub-exponential modulus-to-noise ratio [QWW18], or from ring-LWE with a polynomial modulus [Ros22]. Furthermore, the resulting bilinear HSS and half-chosen VOLE schemes would require two rounds of interaction.

Optimized Constructions From Power-DDH or Power Ring-LWE. The communication of our half-chosen VOLE protocol can further be reduced to $O(n^{1/2})$, by relying on a variant of bilinear HSS for structured

[1] Ignoring factors of $\mathsf{poly}(\lambda)$, for security parameter λ.

matrices with reduced communication cost. This requires either the power-DDH assumption over Paillier groups or class groups, or a new variant of ring-LWE called the *power ring-LWE* assumption.

Power-DDH, introduced by [GJM03], has since been used over both groups and pairing groups, for a variety of applications. Security in the generic group model was proven by [CNs07] (over the target group of a generic pairing), and it is easily shown from generic group model master theorems. When instantiated over Paillier groups or class groups, the hidden subgroup assumption implies that Power-DDH over the full group G is equivalent to Power-DDH over the hidden subgroup H. Other than having unknown order, H behaves similarly to the groups considered in previous works using Power-DDH.

Power Ring-LWE requires pseudorandomness of a set of ring-LWE samples of the form $(a^i, a^i s + e_i)$, where a is a single, public random ring element. The SIS problem dual to this was introduced under the name of "Vanishing SIS" by [CLM23]. However, further research is needed to understand the security of the Power Ring-LWE problem.

Bilinear HSS + HSS for RMS ⇒ succinct HSS for special RMS. We observe that existing constructions of two-party HSS for RMS programs can be upgraded to achieve succinct HSS for special RMS programs, by applying our succinct, half-chosen VOLE protocol. This leads to constructions based on DCR, class groups or LWE (here, we cannot use QR, since there is no suitable HSS for RMS programs based on QR).

Next, we present several applications of succinct HSS.

Batch, multi-party distributed point functions. A point function $f_{\alpha,\beta} : \{0,1\}^k \to \{0,1\}$ is a function, parametrized by α, β, where $f_{\alpha,\beta}(\alpha) = \beta$, and $f_{\alpha,\beta}(x) = 0$ for all $x \neq \alpha$. A distributed point function (DPF) is a way of distributing succinct shares of a secret point function to N parties, such that they can locally compute a share of $f_{\alpha,\beta}(x)$, for any public x.

Using two-party, succinct HSS, we construct a batch, N-party DPF protocol, which distributes shares for m DPFs on a domain of size 2^k, using $O(m+k)+o(m\cdot 2^k)$ communication (ignoring $\mathsf{poly}(\lambda, N)$ factors). Therefore, for sufficiently high m (but still $\mathsf{poly}(\lambda)$), the per-DPF communication becomes $\mathsf{poly}(\lambda)$. If the α, β values defining the point functions are uniformly random, the per-DPF communication even becomes $o(1)$. This effectively gives a pseudorandom correlation generator [BCG+19] for distributing a batch of random DPF instances. This was not known previously, without relying on FHE.

Sublinear MPC without FHE. Finally, we present two new results on MPC with sublinear communication, for any (polynomial) number of parties. The following results are obtained, respectively, via our batch DPF construction and one-time truth tables [Cou19], and directly via succinct HSS.

Theorem 2 (informal). *Let $N = \mathsf{poly}(\lambda)$, and assume either DCR or a DDH-like assumption in class groups. Then, for a sufficiently large Boolean circuit C with s gates, there exists an N-party protocol that securely computes C with semi-honest security and total communication complexity:*

1. *$O(Ns/\log\log s)$, for arbitrary layered circuits C.*
2. *$O(Ns/\log s)$, for layered circuits C that are sufficiently wide.*

For comparison with previous works, the standard, semi-honest GMW protocol with multiplication triples has $O(Ns)$ communication complexity. The result from [BGI16] for 2 parties, using HSS based on DDH, achieved $O(s/\log s)$ complexity for arbitrary layered circuits. Couteau [Cou19] used one-time truth tables to obtain $O(Ns/\log\log s)$ in the correlated randomness model, for any N. The recent work of [DIJL23] achieves $O(Ns/\log\log s)$ complexity for any N, for sufficiently wide, layered circuits using sparse LPN, while [BCM23] achieves $O(s/\log\log s)$ for arbitrary circuits and up to 5 parties.

On malicious security. We present all of our results in the semi-honest security model. Analogous results in the malicious model can be obtained by applying communication-preserving compilers [NN01] based on succinct zero-knowledge arguments, which can be built from collision-resistant hashing [Kil92].

1.2 Technical Overview — Construction of Succinct HSS

Homomorphic secret sharing. Here we briefly review the two-party HSS technique introduced by [BGI16], and subsequently extended in [BKS19, OSY21, RS21, ADOS22]. Suppose that there are secret values x and y, which are unknown to Alice and Bob. However, they both get ciphertexts c and c' encrypting x and y under some public key pk. Let k be the private counterpart of pk. The parties also have subtractive secret shares $[\![k]\!]$ of the secret key k, i.e., Alice has $[\![k]\!]_0$ and Bob has $[\![k]\!]_1$ such that $[\![k]\!]_0 - [\![k]\!]_1 = k$. Alice and Bob want to get secret shares of $x \cdot y$, without communicating.

Some properties are required of the encryption scheme. First, it must be additively homomorphic, i.e., $c \cdot c'$ consists of an encryption of $x+y$. Second, decryption is required to work in a particular way: $k \cdot x = \mathsf{DLog}(c^k)$.[2] Note that this requires some instances of discrete logarithm to be easy. For schemes based on DCR or class groups, there is a subgroup where discrete logarithm is easy, and DLog works within this group. For LWE-based schemes, the equivalent of DLog becomes scaling and rounding, which works whenever the plaintext is only hidden by small noise.

Now, Alice and Bob start by converting the ciphertext c into secret shares, by doing a distributed decryption. They first compute $\langle\!\langle k \cdot x \rangle\!\rangle = c^{[\![k]\!]}$; this is a *multiplicative sharing* of $k \cdot x$, which means that $\mathsf{DLog}(\langle\!\langle k \cdot x \rangle\!\rangle_0 / \langle\!\langle k \cdot x \rangle\!\rangle_1) = k \cdot x$. Next they use a *distributed discrete-log* operation DDLog on their multiplicative shares, which has the property that $\mathsf{DDLog}(\langle\!\langle z \rangle\!\rangle_0) - \mathsf{DDLog}(\langle\!\langle z \rangle\!\rangle_1) = \mathsf{DLog}(\langle\!\langle z \rangle\!\rangle_0 / \langle\!\langle z \rangle\!\rangle_1) = z$,[3] allowing the DLog shares to be computed through purely local computation. They use it to convert $\langle\!\langle k \cdot x \rangle\!\rangle$ into a subtractive sharing $[\![k \cdot x]\!] = \mathsf{DDLog}(\langle\!\langle k \cdot x \rangle\!\rangle)$. This process can then be repeated starting from $[\![k \cdot x]\!]$ (instead of $[\![k]\!]$) and c' (instead of c), to get $\langle\!\langle k \cdot x \cdot y \rangle\!\rangle = c'^{[\![k \cdot x]\!]}$ and $[\![k \cdot x \cdot y]\!] = \mathsf{DDLog}(\langle\!\langle k \cdot x \cdot y \rangle\!\rangle)$. From $[\![k \cdot x \cdot y]\!]$, there are various techniques that allow retrieving $[\![x \cdot y]\!]$. Alice and Bob have now successfully multiplied x by y.

Generalizing this technique gives two-party HSS for Restricted Multiplication Straight-line (RMS) programs. These are arithmetic circuits that support two kinds of values: (a) input ciphertexts (encryptions c of an input x under pk), and (b) memory shares (subtractive secret-sharings $[\![k \cdot x]\!]$ where x is the value and k is the private counterpart of pk); with the restriction that multiplication is only allowed between an input ciphertext and a memory share, but never between two memory shares. In the HSS scheme, multiplications between input ciphertexts and memory wires can be performed without any communication as we outlined above. Additions between memory wires come at essentially no cost thanks to the linearity of subtractive secret sharing.

We recall that RMS programs are powerful enough to implement branching programs and log-depth circuits [BGI16, Appendix A].

Succinct HSS from half-chosen VOLE. While the communication cost of HSS is independent of the circuit size, it still requires sending (encryptions of) both parties' inputs to each other. Even in the worst case, information theory only requires that one party send data proportional to its input size, so we sought to build *succinct HSS*, where some inputs are sent with only sublinear communication. We found a construction of succinct HSS for special RMS programs.

Special RMS programs are RMS programs with two different classes of inputs: Standard inputs are the usual input ciphertexts, and can be multiplied by memory values as normal. Special inputs can only be used as memory values, and can never be multiplied by memory values. A succinct HSS scheme is an HSS scheme for special RMS programs with communication sublinear in the size of the special inputs. We highlight that since HSS supports complex non-linear operations on the input ciphertexts, succinctness for these is impossible in general, by information complexity [Yao83].

In the HSS framework outlined earlier, notice that implementing succinct HSS for a vector of special inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ boils down to succinctly obtaining shares $[\![k \cdot \boldsymbol{x}]\!] = ([\![k \cdot x_1]\!], \ldots, [\![k \cdot x_n]\!])$ of $\boldsymbol{x}$'s product with the secret key. Without loss of generality, suppose that Alice is the party who knows the input $\boldsymbol{x}$. In our construction, she will only send a compact hash of her input, so we will call her the *hasher* and

[2] Technically, this is a simplification for Damgård–Jurik-based HSS [RS21]. For other schemes, $\boldsymbol{k}$ becomes a vector, and c becomes a matrix of group elements which has $\boldsymbol{k}$ as a kind of eigenvector (in the exponent).

[3] For HSS based on DDH, this is only possible with probability $1 - 1/\mathsf{poly}(\lambda)$.

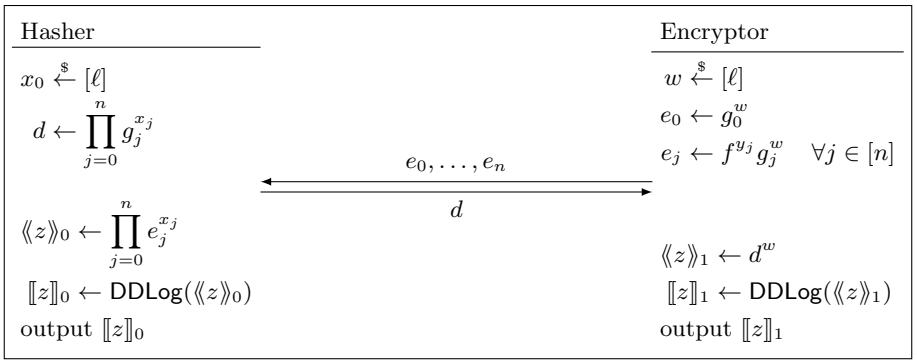

Fig. 1. Bilinear HSS for inner product.

Bob the *encryptor*. Since the parties already have secret shares of $[\![k]\!]$, we have that $[\![k \cdot \boldsymbol{x}]\!] = [\![k]\!]_0 \boldsymbol{x} - [\![k]\!]_1 \boldsymbol{x}$, and the hasher can locally compute the first term $[\![k]\!]_0 \boldsymbol{x}$. The second term, $-[\![k]\!]_1 \cdot \boldsymbol{x}$, is a product of a scalar $-[\![k]\!]_1$ known by the encryptor with a vector $\boldsymbol{x}$ known by the hasher. That is, they need a vector oblivious linear evaluation (VOLE), where the hasher chooses vector $\boldsymbol{x}$ and gets $[\![\Delta \cdot \boldsymbol{x}]\!]_0$, while the encryptor chooses the scalar $\Delta = -[\![k]\!]_1$ and gets $[\![\Delta \cdot \boldsymbol{x}]\!]_1$, such that $[\![\Delta \cdot \boldsymbol{x}]\!]_0 - [\![\Delta \cdot \boldsymbol{x}]\!]_1 = k \cdot \boldsymbol{x}$. Since the hasher only chooses one vector and gets a second vector sampled pseudorandomly, we call this a *half-chosen VOLE*. See Section 7 for more discussion of succinct HSS.

Realizing half-chosen VOLE and bilinear HSS. We consider half-chosen VOLE as a special case of a more general problem, *bilinear HSS*. The hasher and the encryptor have input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively, and they want secret shares of some bilinear function of $\boldsymbol{x}$ and $\boldsymbol{y}$. We additionally require that bilinear HSS has only one simultaneous round, i.e., they must each send some hash or encryption of $\boldsymbol{x}$ and $\boldsymbol{y}$, but further communication (which might depend on both $\boldsymbol{x}$ and $\boldsymbol{y}$) is disallowed. For efficiency, we require that the communication sent by the hasher must be sublinear in their input size.

We start with a bilinear HSS protocol for the dot product $z = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$. Our protocol, illustrated in Fig. 1, is inspired by existing constructions of trapdoor hash functions for linear predicates [DGI+19, RS21]. We work over a group, and assume that the elements $g_0, \ldots, g_n$ each generate the same cyclic subgroup. Moreover, we assume the existence of a subgroup generated by an element f where discrete log is easy. This is consistent with the non-interactive discrete logarithm sharing (NIDLS) framework of [ADOS22], which can be instantiated either from DCR or class groups.

In our protocol, the hasher only sends a Pedersen commitment[4]

$$d = g_0^{x_0} g_1^{x_1} \cdots g_n^{x_n}$$

(the *hash*) to their input $(x_1, \ldots, x_n)$ (x_0 is a random mask), which has size independent of n. Meanwhile, the encryptor picks a random private key w, and sends ElGamal-like ciphertexts $e_j = f^{y_j} g_j^w$ for $j = 0, \ldots, n$ (where $y_0 := 0$). Using the homomorphism on these ciphertexts, they can then get multiplicative shares $\langle\!\langle z \rangle\!\rangle_0 := \prod_{j=0}^{n} e_j^{x_j}$ and $\langle\!\langle z \rangle\!\rangle_1 := d^w$ of $z = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

$$\frac{\langle\!\langle z \rangle\!\rangle_0}{\langle\!\langle z \rangle\!\rangle_1} = \frac{\prod_{j=0}^{n} e_j^{x_j}}{d^w} = \frac{g_0^{w \cdot x_0} \prod_{j=1}^{n} f^{x_j \cdot y_j} g_j^{w \cdot x_j}}{\prod_{j=0}^{n} g_j^{w \cdot x_j}} = \prod_{j=1}^{n} f^{x_j \cdot y_j} = f^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}$$

Finally, DDLog converts these to additive shares $[\![z]\!]$.

Note that Fig. 1 has one simultaneous round, as is required for bilinear HSS. One simultaneous round protocols often have a very useful property: the messages sent by each party can be reused. If the hasher

[4] Note that in any such protocol both parties must at least commit to their inputs. If one didn't, then it could learn two different output shares $[\![z]\!]_i, [\![z']\!]_i$, and take the difference to get $z - z'$ (since the other party's share must be the same both times), which would leak.

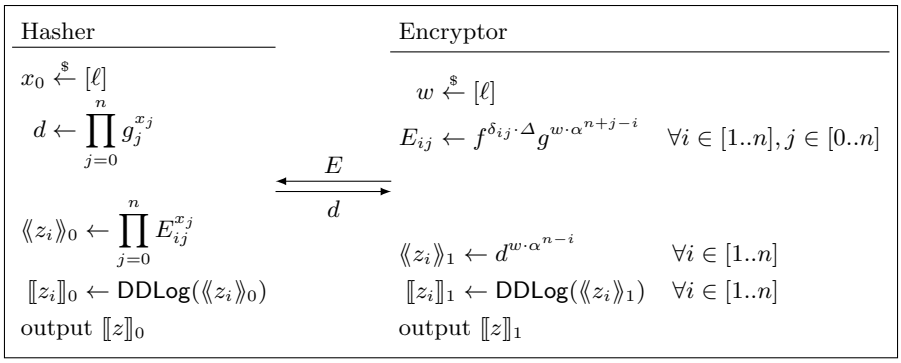

Fig. 2. Bilinear HSS for scalar-vector product from Power DDH. Before this protocol is run, the hasher must receive $g_j = g^{\alpha^j}$ for $j \in [0..n]$ from the encryptor.

generates l messages and the encryptor generates m messages, they can compute secret shares of all $l \cdot m$ inner products. That is, if the hasher's input vectors are the columns of a matrix X and the encryptor's input vectors are the rows of a matrix Y, then they get secret shares of the matrix product $Y \cdot X$. They get all $m \cdot l$ outputs, even though the hasher sends only l group elements and the encryptor sends only $m \cdot (n+1)$ group elements. The output can have significant length, even though we are building on an inner product protocol, which only produces a single output.

Next, we construct bilinear HSS for scalar-vector product with sublinear communication in both directions – i.e., succinct half-chosen VOLE. Let N be the length of the original vector $\boldsymbol{x}$, and let $n = m = N^{1/3}$ and $l = N^{2/3}$. Let $Y = \Delta \cdot \mathsf{id}_n$, where Δ is the scalar input to the VOLE from Bob, and let X be an $n \times l$ matrix such that $\boldsymbol{x}$ is the columns of X stacked together. Then the result is $Y \cdot X = \Delta \cdot X$, and stacking its columns gives $\Delta \cdot \boldsymbol{x}$, so we have a half-chosen VOLE. We have achieved half-chosen VOLE, while sending a total of only $l + m \cdot (n+1) \approx 2 \cdot N^{2/3}$ group elements.

We formally define bilinear HSS in Section 3, detail our initial construction in Section 4.1, and explain how to build succinct half-chosen VOLE in Section 5.

More Efficient Bilinear HSS. In the previous bilinear HSS construction, the hasher had to send an encryption E of the whole matrix Y, even though it was structured as a multiple of the identity matrix. If we could somehow preserve some of this structure in E, then we could compress it.

In Fig. 2, we present such a bilinear HSS construction, based on the Power-DDH assumption used by the efficient range-trapdoor functions of [GHO20]. Here, we describe its relation with Fig. 1. Run Fig. 1 with n repetitions of the encryptor, one for each of the n rows of Y. However, make two changes to reduce the entropy of E, by sampling values in a correlated manner. First, instead of sampling $g_0, \ldots, g_n$ independently, have the encryptor sample a random exponent α, and set $g_j = g^{\alpha^j}$.[5] We require that the g_j's are given to the hasher in an initial setup phase. The Power-DDH assumption states that these are indistinguishable from random group elements. Second, sample the secret keys w_i in a correlated way, as $w_i = w \cdot \alpha^{n-i}$ for random w.

With these two changes, we have

$$E_{ij} = f^{\Delta \cdot \delta_{ij}} g_j^{w_i} = f^{\Delta \cdot \delta_{ij}} g^{w \cdot \alpha^{n+j-i}},$$

where δ_{ij} is the Kronecker delta function (1 if $i = j$, and 0 otherwise). Note that E_{ij} now only depends on $j - i$, so E is constant along diagonals, making it a Toeplitz matrix. Therefore, it can be compressed: the first row and column are enough to reconstruct the rest. This takes $2n$ group elements, as E is an $n \times (n+1)$ matrix.

[5] This requires some Setup to be run before the protocol starts, since the hasher must not learn α.

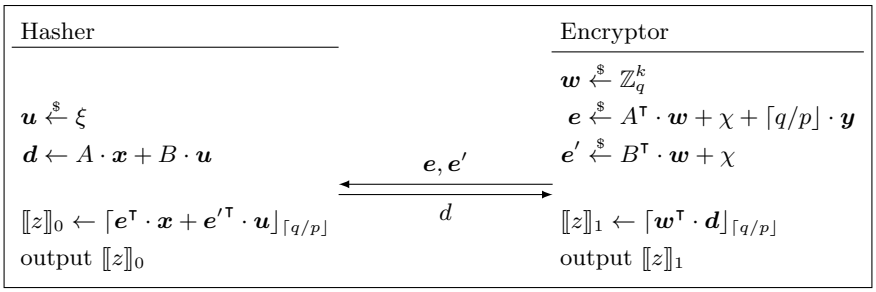

Fig. 3. Bilinear HSS for inner product (mod p) from LWE. Here $A \in \mathbb{Z}_q^{k \times n}, B \in \mathbb{Z}_q^{k \times t}$ are public random matrices, and ξ and χ are noise distributions.

Correctness still works in the same way:

$$\frac{\langle\!\langle z_i \rangle\!\rangle_0}{\langle\!\langle z_i \rangle\!\rangle_1} = \frac{\prod_{j=0}^n E_{ij}^{x_j}}{d^{w \cdot \alpha^{n-i}}} = \frac{\prod_{j=0}^n f^{\delta_{ij} \cdot \Delta \cdot x_j} g^{w \cdot \alpha^{n+j-i} \cdot x_j}}{\prod_{j=0}^n g_j^{w \cdot \alpha^{n-i} \cdot x_j}} = f^{\Delta \cdot x_i} \cdot \frac{\prod_{j=0}^n g^{w \cdot \alpha^{n+j-i} \cdot x_j}}{\prod_{j=0}^n g^{w \cdot \alpha^{n+j-i} \cdot x_j}} = f^{\Delta \cdot x_i}$$

After DDLog, the protocol outputs additive shares of the vector $\Delta \cdot \boldsymbol{x}$.

Now, we can further improve efficiency by reusing E for many different hashes. Divide $\boldsymbol{x}$ into $N^{1/2}$ blocks $\boldsymbol{x}_k$ of $n = N^{1/2}$ elements each, then compute shares of $\Delta \cdot \boldsymbol{x}_k$ for all k to get $\Delta \cdot \boldsymbol{x}$. The communication cost is $O(N^{1/2})$: $N^{1/2}$ hashes, plus a single matrix E containing $2 \cdot N^{1/2}$ group elements. We detail this construction in Section 6.1.

LWE-Based Bilinear HSS. We also designed protocols for bilinear HSS from the learning with errors (LWE) assumption. In Fig. 3 we present a direct translation of Fig. 1 from groups to lattices. Assume that $\boldsymbol{x} \in \mathbb{Z}_p^n$ and $\boldsymbol{y} \in \mathbb{Z}_p^n$ are encoded into the larger ring $\mathbb{Z}_q$ by taking the smallest representative modulo p. Then the digest becomes the usual SIS hash function: $\boldsymbol{d} = A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u}$ [Ajt96], for public random matrices $A \in \mathbb{Z}_q^{k \times n}, B \in \mathbb{Z}_q^{k \times t}$, and a small random mask $\boldsymbol{u}$. The encryption $\boldsymbol{e}, \boldsymbol{e'}$ of y becomes a dual-Regev-like ciphertext

$$\boldsymbol{e} = A^\intercal \cdot \boldsymbol{w} + \chi + \lceil q/p \rfloor \cdot \boldsymbol{y}, \qquad \boldsymbol{e'} = B^\intercal \cdot \boldsymbol{w} + \chi,$$

where χ is a noise distribution [GPV08].

For correctness, first notice that before rounding the output shares are nearly correct, assuming that the noise distribution is sufficiently small.

$$\begin{aligned} \boldsymbol{e}^\intercal \cdot \boldsymbol{x} + \boldsymbol{e}'^\intercal \cdot \boldsymbol{u} - \boldsymbol{w}^\intercal \cdot \boldsymbol{d} &= (\boldsymbol{w}^\intercal \cdot A + \chi^\intercal + \lceil q/p \rfloor \cdot \boldsymbol{y}^\intercal) \cdot \boldsymbol{x} + (\boldsymbol{w}^\intercal \cdot B + \chi^\intercal) \cdot \boldsymbol{u} \\ &\quad - \boldsymbol{w}^\intercal \cdot (A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u}) \\ &= \lceil q/p \rfloor \cdot \boldsymbol{y}^\intercal \cdot \boldsymbol{x} + \chi^\intercal \cdot \boldsymbol{x} + \chi^\intercal \cdot \boldsymbol{u} \\ &\approx \lceil q/p \rfloor \cdot (\langle \boldsymbol{x}, \boldsymbol{y} \rangle \bmod p) \end{aligned}$$

Next, if this pre-rounding correctness error is negligible compared to q/p, after rounding, the result will be correct with all but negligible probability:

$$[\![z]\!]_0 - [\![z]\!]_1 = \lceil \boldsymbol{e}^\intercal \cdot \boldsymbol{x} + \boldsymbol{e}'^\intercal \cdot \boldsymbol{u} \rfloor_{\lceil q/p \rfloor} - \lceil \boldsymbol{w}^\intercal \cdot \boldsymbol{d} \rfloor_{\lceil q/p \rfloor} \equiv \langle \boldsymbol{x}, \boldsymbol{y} \rangle \mod p$$

The communication cost is asymptotically the same as for Fig. 1, with each hash being k elements of $\mathbb{Z}_q$, and each vector y being $n + t$ elements of $\mathbb{Z}_q$. We detail this construction in Section 4.2.

Power Ring-LWE. We also adapted the ideas from Fig. 2 to work with LWE. Our protocol is Fig. 4. Again, the main idea of the protocol is structure the ciphertexts so that E becomes a Toeplitz matrix, and so can be compressed. To support this additional structure, we need to define a new security assumption: Power Ring-LWE. In standard Ring-LWE, the adversary is given as many samples of the form $(a_i, a_i \cdot s + e_i)$ as they

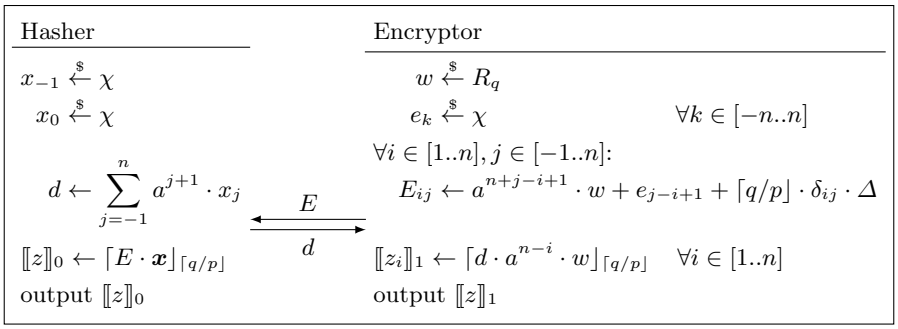

Fig. 4. Bilinear HSS for scalar-vector product from Power Ring-LWE. Here $a \in R_q$ is a public ring element, and χ is a noise distribution on R_q.

want, where s and all a_i are uniform and e_i is from a noise distribution χ, and they must distinguish from random. Power Ring-LWE is similar, but the public ring elements a_i are structured: they satisfy $a_i = a^i$. In this way, the digest value d becomes a polynomial evaluation[6] at a: $d = \sum_{j=-1}^{n} a^{j+1} x_j$. Note that some random values have been prepended to $\boldsymbol{x}$ to hide the hasher's input.

Also like with Fig. 2, the encryptor's secret keys w_i values are correlated by $w_i = a^{n-i} \cdot w$. Ignoring noise, this makes the ciphertexts become

$$E_{ij} \approx a^{n+j-i+1} \cdot w + \lceil q/p \rfloor \cdot \delta_{ij} \cdot \Delta,$$

which again only depends on $j-i$, so it is a Toeplitz matrix. For negligible noise-to-modulus ratio, correctness holds because

$$\begin{aligned}
\Big(\sum_{j=-1}^{n} E_{ij} \cdot x_j \Big) - d \cdot a^{n-i} \cdot w &= \sum_{j=-1}^{n} \big(a^{n+j-i+1} \cdot w + e_{j-i+1} + \lceil q/p \rfloor \cdot \delta_{ij} \cdot \Delta \big) \cdot x_j \\
&\quad - \sum_{j=-1}^{n} a^{n+j-i+1} \cdot x_j \cdot w \\
&= \lceil q/p \rfloor \cdot \Delta \cdot x_i + \sum_{j=-1}^{n} e_{j-i+1} \cdot x_j \\
&\approx \lceil q/p \rfloor \cdot (\Delta \cdot x_i \bmod p).
\end{aligned}$$

The communication cost is similar to Fig. 2, with the hasher sending a single ring element, and the encryptor sending $2n + 1$ ring elements: the first row and first column of E. We detail this construction in Section 6.2.

1.3 Technical Overview — Applications of Succinct HSS

One main application of HSS is to MPC with sublinear communication [BGI16]. Alice and Bob divide their circuit into layers of logarithmic depth, and use HSS to evaluate each layer. Communication is only required between layers, so they only send $O(s/\log s)$ bits for a circuit with s gates.

The biggest limitation of this technique is that it only applies to two parties, since the underlying HSS only works for two parties. Recent work has addressed this gap by building truth tables for $(\log \log s)$-depth circuit blocks, which have at most $\log(s)$ inputs and so have polynomial-sized truth tables. [Cou19] showed that if the parties are given correlated randomness in the form of (batched) One-Time Truth-Tables (OTTTs)[7]

[6] The SIS problem corresponding to d being a collision-resistant hash was defined in [CLM23]. They named it vanishing SIS, and gave some evidence for its hardness.

[7] An OTTT [IKM+13] is a truth table, permuted according to a random input mask, secret shared among all N parties.

for these blocks, then they can evaluate the layered circuit with $O(s/\log\log s)$ communication. However, they were unable to efficiently generate these OTTTs without using assumptions that imply FHE. [BCM23] got past this issue by generating truth tables for two parties with HSS, then evaluating them on the third party's input using a variant of PIR. Their technique can be stretched to work with 5 parties, but no further. Recently, another work [DIJL23] managed to achieve $O(s/\log\log s)$ communication — when the circuit is sufficiently wide — for any number of parties, based on a sparse variant of the learning parity with noise assumption.

Sublinear MPC from succinct HSS. We show how to generate these OTTTs with only sublinear communication, using succinct HSS. We consider the following procedure to generate a OTTT of a function $f\colon \{0,1\}^k \to \{0,1\}$.

1. Party $\mathcal{P}_1$ generates a random mask $r_1 \in \{0,1\}^k$, and computes their permuted truth table $s_1[x] \leftarrow f(x \oplus r_1)$ for all $x \in \{0,1\}^k$.
2. For $j = 2, \ldots, N$, party $\mathcal{P}_j$ generates a random mask $r_j \in \{0,1\}^k$, and uses succinct HSS to permute the truth table as follows:
   (a) Currently, we have that $\sum_{i=1}^{j-1} s_i[x] = f(x \oplus r_1 \oplus \cdots \oplus r_{j-1})$.
   (b) For every $i < j$, parties $\mathcal{P}_i$ and $\mathcal{P}_j$ run 2-party succinct HSS on input the truth table $\boldsymbol{s_i}$ and the mask r_j. They compute secret shares $s'_{ij}[x]$ and $s'_{ji}[x]$ of $s_i[x \oplus r_j]$ for all x, so $s'_{ij}[x] - s'_{ji}[x] = s_i[x \oplus r_j]$. Party $\mathcal{P}_i$ updates their share $s_i \leftarrow s'_{ij}$.
   (c) Party $\mathcal{P}_j$ sets their share to $s_j \leftarrow -\sum_{i=1}^{j-1} s'_{ji}$.
   (d) Now $\displaystyle\sum_{i=1}^{j} s_i[x] = \sum_{i=1}^{j-1} s'_{ij}[x] - \sum_{i=1}^{j-1} s'_{ji}[x] = f\Big(x \oplus r_1 \oplus \ldots \oplus r_{j-1} \oplus r_j\Big)$.
3. Each party $\mathcal{P}_i$ outputs s_i as their share of the OTTT. The loop invariant guarantees that $\sum_{i=1}^{N} s_i[x] = f(x \oplus r)$, where $r = \bigoplus_{i=1}^{N} r_i$, so these shares form a OTTT for f.

This protocol can also be batched to generate many OTTTs at once. If the underlying HSS has sublinear communication complexity in its inputs then this protocol can generate the OTTTs needed by [Cou19] with communication sublinear in the circuit size, completing their sublinear MPC protocol.

We notice that in all HSS evaluation, the input s_i is only used linearly, as the circuit is only permuting the entries of s_i according to $x \mapsto x \oplus r_j$. In other words, given succinct HSS, s_i can be compressed. Unfortunately, the random mask r_j is used non-linearly as it defines the permutation on s_i, so it cannot be compressed in the same way. However, it is sampled randomly, so for each j, all the r_j's across all OTTTs can be compressed together using a PRF in NC_1. Since all inputs can be compressed, we achieve sublinear communication MPC. We detail this construction in Section 8.1.

Multiparty DPFs. In the special case of a OTTT for the function f, given by $f(0) = \beta$ and otherwise $f(x) = 0$, we get a protocol for generating a batch of multiparty DPFs using sublinear communication (the OTTT mask r corresponds to the non-zero point of the point function). Most directly, this would require giving β to $\mathcal{P}_1$, but with a minor tweak to the protocol, β can be secret shared among the N parties instead. Note that the communication complexity is polynomial in the number of parties, and that for a sufficiently large batch size the per-DPF communication can be made arbitrarily small. We detail this construction in Section 8.

Sublinear MPC for RMS programs. Another application of succinct HSS is to (interactively) evaluate layered RMS programs in MPC for a polynomial number of parties, with communication sublinear in the program size. A layered RMS program allows arbitrary fan-in addition gates and 2-input multiplication gates, with the restriction that every gate must output into a later layer than any of its inputs.

Our protocol for evaluating layered RMS programs works as follows. Maintain additive secret shares $y_1 + \cdots + y_N$ of every memory value in the RMS program. We can initially share the inputs as well, since $(0, \ldots, x, \ldots, 0)$ is an additive sharing of x, and every input is known by some party. We can easily evaluate

addition gates directly on the shares, just by adding the shares. The difficulty comes when we need to multiply an input x by some memory value y. We implement multiplications with succinct HSS, by having the shares of y be compressed as special inputs.

In more detail, to handle the multiplications for each layer, we run a succinct HSS evaluations between every pair of parties. If party $\mathcal{P}_i$ has an input x_i, it provides x_i as a standard input to all of its succinct HSS evaluations.[8] Now, every multiplication in this layer can be handled as follows. If we need to evaluate $x \cdot y$, for an input x held by party $\mathcal{P}_i$ and a memory value $y = y_1 + \cdots + y_N$:

1. For all $j \neq i$, include y_j as a special input (if it is not already included) to the succinct HSS between $\mathcal{P}_i$ and $\mathcal{P}_j$, and compute shares of $x \cdot y_j$.
2. $\mathcal{P}_i$ locally evaluates $x \cdot y_i$.
3. Sum the above shares to get shares of $x \cdot y_1 + \cdots + x \cdot y_N = x \cdot y$.

Note that all multiplications in a single layer use the same succinct HSS, so that all of these share inputs get compressed together in the succinct HSS. Therefore, for efficiency we need a sufficiently wide RMS program. Note also that the round complexity is proportional to the number of layers.

We can also use this protocol to get sublinear complexity MPC for layered circuits, as long as the layers are wide enough and even in size. Similarly to [BGI16], divide the circuit into blocks of depth $c \cdot \log s$ for some $0 < c < 1$, and convert each block into an $o(s)$-layer RMS program. Evaluate each block using the above protocol for RMS programs, and input the shares output from each layer into the next. For efficiency, however, we require hybrid encryption for the inputs, so the parties will instead generate symmetric-key ciphertexts from their shares, and input $\log(\lambda)$-depth PRF keys to the RMS program so that they can be decrypted. This increases the circuit depth to $c \cdot \log s + O(\log(\lambda))$, and so the RMS program will have $s^c \mathsf{poly}(\lambda)$ layers.

For sufficiently wide circuits, the symmetric-key ciphertexts become the biggest term asymptotically, and so the whole circuit can be evaluated with $O(s/\log s)$ communication. However, this construction blows up the round complexity to $d \cdot \frac{s^c \mathsf{poly}(\lambda)}{c \log s}$, if d is the depth of the original circuit. We detail this construction in Section 9.

2 Notation and Preliminaries

We denote the security parameter by λ. For any $n \in \mathbb{N}$, we define $[n] := \{1, \ldots, n\}$ and $[0..n] := \{0, 1, \ldots, n\}$. For any $x, N \in \mathbb{N}$ where N is odd, we use $\left(\frac{x}{N}\right)$ to denote the Jacobi symbol of x and N. Given a real number x and an integer p, we use $\lceil x \rfloor_p$ to denote the integer y such that $y \cdot p$ is the multiple of p that is closest to x. We use $\lceil x \rfloor$ to denote rounding to the closest integer.

Objects in vectorial form are represented using bold font. For any vector $\boldsymbol{v}$, we denote the i-th entry either by $\boldsymbol{v}[i]$ or by $\boldsymbol{v}_i$. Matrices are represented using capital letters. The element in the i-th row and j-th column of a matrix A is denoted by $A_{i,j}$. The transposition of a matrix A is denoted by $A^\intercal$. Unit-vectors are vectors where all the entries are zero except for, perhaps, one. We call the latter the *special position* or the *special entry*. The corresponding value is called the *non-zero element*. We denote a unit vector with special position α and non-zero element β by $\boldsymbol{u}_{\boldsymbol{\alpha},\boldsymbol{\beta}}$. We use $\otimes$ to denote the outer product.

For any randomised algorithm Alg, we use $y \xleftarrow{\$} \mathsf{Alg}(x)$ to mean that y is assigned the output of Alg on input x and uniformly random coins. We use instead $y \leftarrow \mathsf{Alg}(x; \boldsymbol{r})$ to mean that y is assigned the output of Alg on input x and randomness $\boldsymbol{r}$. If Alg is deterministic, we simply write $y \leftarrow \mathsf{Alg}(x)$. Finally, for any finite set S, we write $y \xleftarrow{\$} S$ if y is assigned the value of a uniformly random element in S.

Throughout the paper, we deal with multiparty computation protocols. We denote the number of parties by N. The i-th party is denoted by $\mathcal{P}_i$. In all protocols, we assume static, semi-honest corruption.

[8] These input ciphertexts can sent to all parties at the start, and reused for all layers.

2.1 Standard Assumptions

Below, we recall some basic computational assumptions we use throughout the paper. We start from the quadratic residuosity (QR) assumption over the RSA group. The latter states that given a product of random safe-primes N (where the factorisation is kept secret), it is hard to distinguish between a random square in $\mathbb{Z}_N^*$ and a random element in $\mathbb{Z}_N^*$ having Jacobi symbol equal to 1. We recall that a safe prime is a prime number $p = 2p' + 1$ where p' is also prime.

Definition 1 (Quadratic Residuosity over the RSA Group). *We say that the quadratic residuosity (QR) assumption holds over the RSA group, if the following distributions are computationally indistinguishable*

$$\left\{ g, N \;\middle|\; \begin{array}{l} p, q \xleftarrow{\$} \textit{random } \lambda\textit{-bit safe-primes} \\ N \leftarrow p \cdot q \\ g \xleftarrow{\$} \left\{ x \in \mathbb{Z}_N^* \;\middle|\; \left(\frac{x}{N}\right) = 1 \right\} \end{array} \right\}$$

$$\left\{ g^2, N \;\middle|\; \begin{array}{l} p, q \xleftarrow{\$} \textit{random } \lambda\textit{-bit safe-primes} \\ N \leftarrow p \cdot q \\ g \xleftarrow{\$} \mathbb{Z}_N^* \end{array} \right\}$$

We now recall the decisional composite residuosity (DCR) assumption over the Paillier group. We recall that the Paillier group is $\mathbb{Z}_{N^2}^*$ where N is the product of two random safe-primes. The DCR assumption states that over the Paillier group (where the factorisation of N is kept secret), it is hard to distinguish between a random N-th power and a random element of the group.

Definition 2 (Decisional Composite Residuosity). *We say that the decisional composite residuosity (DCR) assumption holds over the Paillier group, if the following distributions are computationally indistinguishable*

$$\left\{ g, N \;\middle|\; \begin{array}{l} p, q \xleftarrow{\$} \textit{random } \lambda\textit{-bit safe-primes} \\ N \leftarrow p \cdot q \\ g \xleftarrow{\$} \mathbb{Z}_{N^2}^* \end{array} \right\}$$

$$\left\{ g^N, N \;\middle|\; \begin{array}{l} p, q \xleftarrow{\$} \textit{random } \lambda\textit{-bit safe-primes} \\ N \leftarrow p \cdot q \\ g \xleftarrow{\$} \mathbb{Z}_{N^2}^* \end{array} \right\}$$

Finally, we recall the learning with errors (LWE) assumption [Reg05]. The latter states that, given a random lattice described by a thin matrix A over $\mathbb{Z}_q$, if we perturb a random point of the lattice with some noise, the result looks like a random point of the space.

Definition 3 (Learning with Errors). *Let χ_σ^M denote a discrete Gaussian distribution over $\mathbb{Z}^M$ with noise parameter σ. We say that the learning with errors (LWE) assumption holds for the parameters $N(\lambda), M(\lambda), q(\lambda), \alpha(\lambda)$ if, for $\sigma := \alpha(\lambda) \cdot q(\lambda)$, the following distributions are computationally indistinguishable*

$$\left\{ A, A \cdot \boldsymbol{s} + \boldsymbol{e} \;\middle|\; \begin{array}{l} A \xleftarrow{\$} \mathbb{Z}_q^{M \times N} \\ \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^N \\ \boldsymbol{e} \xleftarrow{\$} \chi_\sigma^M \end{array} \right\}$$

$$\left\{ A, \boldsymbol{v} \;\middle|\; \begin{array}{l} A \xleftarrow{\$} \mathbb{Z}_q^{M \times N} \\ \boldsymbol{v} \xleftarrow{\$} \mathbb{Z}_q^M \end{array} \right\}$$

2.2 Assumptions in the NIDLS Framework

Some of our bilinear HSS constructions are based on the non-interactive discrete logarithm sharing (NIDLS) framework of [ADOS22]. We recall here its definition and some known instantiations.

The NIDLS framework consists of a finite commutative group G that can be decomposed as the direct product $F \times H$. The subgroup F is cyclic of known order. Furthermore, it is easy to solve discrete logarithms with respect to a generator f. The subgroup H is instead of unknown order and computing discrete logarithms over it is hard. The framework is also equipped with an upper-bound on the order of G, which we denote by ℓ, and a distribution $\mathcal{D}$ that provides (non-necessarily uniformly) random elements in G.

The most interesting property of the framework is that it allows compute discrete logarithms in a distributed way: if two parties $\mathcal{P}_0$ and $\mathcal{P}_1$ hold elements g_0 and g_1 such that $g_0 = f^m \cdot g_1$ for some $m \in \mathbb{N}$, the parties can derive a subtractive secret-sharing of m without having to interact. The operation is called *distributed discrete logarithm* or DDLog.

Definition 4 (The NIDLS Framework [ADOS22]). *The NIDLS framework consists of a triple of PPT algorithms* $(\mathsf{Gen}, \mathcal{D}, \mathsf{DDLog})$ *with the following syntax:*

- $\mathsf{Gen}(1\!\!1^\lambda)$ *outputs a tuple* $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux})$ *where*
  - G *is a finite abelian group*
  - F *and* H *are subgroups of* G *such that* $G = F \times H$
  - $F = \langle f \rangle$ *and* $|F| = q$
  - ℓ *is a positive integer*
  - aux *consists of auxiliary information*
- $\mathcal{D}(1\!\!1^\lambda, \mathsf{par})$ *outputs an element* $g \in G$ *along with auxiliary information* ρ
- $\mathsf{DDLog}(\mathsf{par}, g)$ *is deterministic and outputs an element* $s \in \mathbb{Z}_q$.

We additionally require the following properties:

- *For every PPT adversary* $\mathcal{A}$,

$$\Pr\left[s_0 - s_1 \not\equiv m \bmod q \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g_0, m) \xleftarrow{\$} \mathcal{A}(1\!\!1^\lambda, \mathsf{par}) \\ g_1 \leftarrow f^m \cdot g_0 \\ s_0 \leftarrow \mathsf{DDLog}(\mathsf{par}, g_0) \\ s_1 \leftarrow \mathsf{DDLog}(\mathsf{par}, g_1) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

- *The following distributions are statistically indistinguishable*

$$\left\{ \mathsf{par}, g, \rho, g^r \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ r \xleftarrow{\$} [\ell] \end{array} \right\}$$

$$\left\{ \mathsf{par}, g, \rho, h \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ h \xleftarrow{\$} \langle g \rangle \end{array} \right\}$$

2.3 Known Instantiations of the Framework.

We now recall the known instantiations of the framework: the Paillier group (and its generalisation Damgård-Jurik) [OSY21,RS21], Goldwasser-Micali (and some variants of its generalisation Joye-Libert) [ADOS22] and class groups [ADOS22].

The Paillier instantiation. We consider the subgroup G of squares of $\mathbb{Z}^*_{N^2}$. Such subgroup is cyclic of order $N \cdot \phi(N)/4$ where $\phi(N)$ denotes Euler's totient function. It is possible to decompose G as the direct product of $F := \langle 1 + N \rangle$ and H, the subgroup of $2N$-th powers of $\mathbb{Z}^*_{N^2}$. It is easy to observe that $(1 + N)^a \equiv 1 + a \cdot N \bmod N^2$ for any $a \in \mathbb{N}$. From this we can easily conclude that the order of F is N and discrete logarithms are easy compute. We refer to [ADOS22] for a detailed discussion on how the distributed DLOG can be computed over Paillier. In Fig. 5, we present a formal description of the Paillier instantiation of the framework (see $\mathsf{Gen}_{\mathsf{Paillier}}$ and $\mathcal{D}_{\mathsf{Paillier}}$).

The Goldwasser-Micali instantiation. Let N be the product of two large random safe-primes. We consider the subgroup G of all elements in $\mathbb{Z}^*_N$ having Jacobi symbol equal to 1. Such group is cyclic of order $\phi(N)/4$. It is possible to decompose G as the product of $F = \{1, -1\}$ and H, the subgroup of squares of $\mathbb{Z}^*_N$. Due to the size, discrete logarithms over F are trivially to solve. The same holds for distributed DLOGs (see [ADOS22]). In Fig. 5, we present a formal description of the Goldwasser-Micali instantiation of the framework (see $\mathsf{Gen}_{\mathsf{GM}}$ and $\mathcal{D}_{\mathsf{GM}}$).

The class group instantiation. The last known instantiation of the framework is given by class groups. Class groups are commutative groups for which it is computationally expensive to compute the order. One of the reasons why class groups are popular in cryptography is the fact that it is possible to generate the parameters of the group with a transparent setup: while the security of Paillier and Goldwasser-Micali requires that the factorisation of N is kept secret, the computational assumptions on class groups remain solid even if we leak the random coins used to sample the group ($\mathsf{Gen}_{\mathsf{CL}}$ will provide them as part of aux).

Any class groups G can be decomposed as the product of F and H. The subgroup F is generated by a group element f of prime order q (such q does not need to be sampled at random, it can be given as input to $\mathsf{Gen}_{\mathsf{CL}}$). Computing DLOGs over F is easy. The subgroup H has instead unknown order and, in most cases, it is not even cyclic. The sampling procedure $\mathcal{D}_{\mathsf{CL}}$ can be instantiated in different ways: it could either be a distribution that is statistically close to uniform over G or it could be a distribution producing random (but not uniformly random) elements of high unknown order. In both cases $\mathcal{D}_{\mathsf{CL}}$ outputs the random coins it uses as part of the auxiliary information ρ. We refer to [ADOS22] for a detailed discussion on how the distributed DLOG can be computed over class groups.

Below, we recall the computational assumptions that are believe to hold in all known instantiations of the NIDLS framework [ADOS22].

We start from the hidden subgroup assumption, which states that, given the description of the NIDLS group and a sample (g, ρ) from $\mathcal{D}$, it is difficult to distinguish between a random power g^r and $f^s \cdot g^r$ for a random $s \in \mathbb{Z}_q$.

Definition 5 (Hidden Subgroup Assumption). *We say that the hidden subgroup assumption holds in the NIDLS framework if the following distributions are computationally indistinguishable*

$$\left\{ \mathsf{par}, g, \rho, f^s \cdot g^r \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ s \xleftarrow{\$} \mathbb{Z}_q \\ r \xleftarrow{\$} [\ell] \end{array} \right\}$$

$$\left\{ \mathsf{par}, g, \rho, g^r \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ r \xleftarrow{\$} [\ell] \end{array} \right\}$$

We then recall the small exponent assumption. The latter states that, for a random sample (g, ρ) from $\mathcal{D}$, the power g^r for $r \xleftarrow{\$} [2^\lambda]$ looks like a random element in $\langle g \rangle$.

Instantiations of the NIDLS Framework over Paillier and Goldwasser-Micali

$\mathsf{Gen}_{\mathsf{Paillier}}(1\!\!1^\lambda)$

1. Sample random λ-bit safe-primes p and q
2. $N \leftarrow p \cdot q$
3. $G \leftarrow \{g^2 | g \in \mathbb{Z}^*_{N^2}\}$
4. $H \leftarrow \{g^N | g \in \mathbb{Z}^*_{N^2}\}$
5. $f \leftarrow 1 + N$
6. $q \leftarrow N$
7. $\ell \leftarrow 2^\lambda \cdot N^2$
8. $F \leftarrow \langle 1 + N \rangle$
9. Output $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} := N)$

$\mathcal{D}_{\mathsf{Paillier}}(1\!\!1^\lambda, \mathsf{par})$

1. $\boldsymbol{\rho} \xleftarrow{\$} \mathbb{Z}^*_{N^2}$
2. Output $(\rho^2, \boldsymbol{\rho})$

$\mathsf{Gen}_{\mathsf{GM}}(1\!\!1^\lambda)$

1. Sample random λ-bit safe-primes p and q
2. $N \leftarrow p \cdot q$
3. $G \leftarrow \left\{x \in \mathbb{Z}^*_N \;\middle|\; \left(\frac{x}{N}\right) = 1\right\}$
4. $H \leftarrow \{g^2 | g \in G\}$
5. $f \leftarrow -1$
6. $q \leftarrow 2$
7. $\ell \leftarrow N$
8. $F \leftarrow \{1, -1\}$
9. Output $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} := N)$

$\mathcal{D}_{\mathsf{GM}}(1\!\!1^\lambda, \mathsf{par})$

1. $x \xleftarrow{\$} \left\{x \in \mathbb{Z}^*_N \;\middle|\; \left(\frac{x}{N}\right) = 1\right\}$
2. Output $(x, \rho := x)$

Fig. 5. Instantiations of the NIDLS Framework over Paillier and Goldwasser-Micali

Definition 6 (Small Exponent Assumption). *We say that the small exponent assumption holds in the NIDLS framework if the following distributions are computationally indistinguishable*

$$\left\{ \mathsf{par}, g, \rho, g^r \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ r \xleftarrow{\$} [\ell] \end{array} \right\}$$

$$\left\{ \mathsf{par}, g, \rho, g^r \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ r \xleftarrow{\$} [2^\lambda] \end{array} \right\}$$

Finally, we formalise adaptations of DDH and power-DDH to the NIDLS framework. Essentially, these correspond to the usual formulation of the assumptions, but we also provide the adversary with a description of the NIDLS group parameters and the auxiliary information ρ output by $\mathcal{D}$ (which is used to sample the base group element g).

Definition 7 (Decisional Diffie-Hellman). *We say that the decisional Diffie-Hellman (DDH) assumption holds in the NIDLS framework if the following distributions are computationally indistinguishable*

$$\left\{ \mathsf{par}, \rho, g, g^a, g^b, g^{a \cdot b} \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ a, b \xleftarrow{\$} [\ell] \end{array} \right\}$$

$$\left\{ \mathsf{par}, \rho, g, g^a, g^b, g^c \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ a, b, c \xleftarrow{\$} [\ell] \end{array} \right\}$$

Definition 8 (Power-DDH). *We say that the n-ary Power-DDH assumption holds in the NIDLS framework if the following distributions are computationally indistinguishable*

$$\left\{ \mathsf{par}, \rho, g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^n} \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ \alpha \xleftarrow{\$} [\ell] \end{array} \right\}$$

$$\left\{ \mathsf{par}, \rho, g, g^{\alpha_1}, g^{\alpha_2}, \ldots, g^{\alpha_n} \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ \forall i \in [n] : \alpha_i \xleftarrow{\$} [\ell] \end{array} \right\}$$

3 Defining Bilinear HSS

In this section, we provide formal definitions of bilinear HSS, describing its syntax and properties.

A bilinear HSS scheme consists of a 2-party primitive: one party is called the hasher and provides as input an n-dimensional vector $\boldsymbol{x}$, the other party is called the encryptor and provides as input a matrix M. The primitive relies on a setup that takes as input n and distributes keys to the parties. Such keys allow the hasher and the encryptor to obtain a secret-sharing of $M \cdot \boldsymbol{x}$ using a single round of interaction. Furthermore, the digest sent by the hasher will have sublinear size in n.

Definition 9 (Bilinear HSS). *Let R be a commutative ring. An R-bilinear HSS scheme for the matrix class $\mathcal{M}$ is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Matrix}, \mathsf{HasherEval}, \mathsf{MatrixEval})$ *with the following syntax:*

– Setup *is randomised and takes as input the security parameter* $\mathbb{1}^\lambda$ *and the input length* $\mathbb{1}^n$*. The output is a hasher key* hk *and a matrix key* mk*.*
– Hash *is randomised and takes as input a hasher key* hk *and an input* $\boldsymbol{x} \in R^n$*. The output is a digest* d *and the hasher secret information* ψ*.*
– Matrix *is randomised and takes as input a matrix key* mk *and an* n*-column matrix* $M \in \mathcal{M}$ *of elements in* R*. The output is an encoding* E *and the matrix secret information* ϕ*.*
– $\mathsf{HasherEval}$ *is deterministic and takes as input a hasher key* hk*, a matrix encoding* E *and hasher secret information* ψ*. The output is a vector* $\boldsymbol{s_0}$*.*
– $\mathsf{MatrixEval}$ *is deterministic and takes as input a matrix key* mk*, a digest* d *and matrix secret information* ϕ*. The output is a vector* $\boldsymbol{s_1}$*.*

We require also the following properties

1. ***(Correctness).*** *For every* $n \in \mathbb{N}$*,* $\boldsymbol{x} \in R^n$ *and* n*-column matrix* $M \in \mathcal{M}$*, we have*

$$\Pr\left[\boldsymbol{s_0} - \boldsymbol{s_1} \neq M \cdot \boldsymbol{x} \;\middle|\; \begin{array}{l} (\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ (d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, \boldsymbol{x}) \\ (E, \phi) \xleftarrow{\$} \mathsf{Matrix}(\mathsf{mk}, M) \\ \boldsymbol{s_0} \leftarrow \mathsf{HasherEval}(\mathsf{hk}, E, \psi) \\ \boldsymbol{s_1} \leftarrow \mathsf{MatrixEval}(\mathsf{mk}, d, \phi) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2. ***(Hasher Privacy).*** *No PPT adversary* $\mathcal{A}$ *can win the game in Fig. 6 with non-negligible advantage.*

THE GAME $\mathcal{G}_n^{\mathsf{HP}}(\lambda)$

Initialisation: This procedure is run only once at the beginning of the game.
(a) $b \xleftarrow{\$} \{0,1\}$
(b) Activate $\mathcal{A}$ with $\mathbb{1}^\lambda$ and receive n in unary notation.
(c) $(\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$
(d) Provide mk to the adversary.
Query: This procedure can be run multiple times and at any moment.
(a) Receive $\boldsymbol{x} \in R^n$ from $\mathcal{A}$.
(b) $(d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, \boldsymbol{x})$
(c) Provide the adversary with d.
Challenge: This procedure can be run only once and at any moment.
(a) Receive $\boldsymbol{x_0}, \boldsymbol{x_1} \in R^n$ from $\mathcal{A}$.
(b) $(d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, \boldsymbol{x_b})$
(c) Provide the adversary with d.
Win: The adversary wins if it ends its execution outputting b.

Fig. 6. The Game for Hasher Privacy

3. ***(Matrix Privacy).*** *No PPT adversary* $\mathcal{A}$ *can win the game in Fig. 7 with non-negligible advantage.*
4. ***(Efficiency).*** *The size of the hash* d *is* $\mathsf{poly}(\lambda, \log|R|) \cdot o(n)$*.*

Notice that hasher privacy is essentially saying that the digests sent by the hasher reveal no information about the underlying messages, even if they are all generated using the same hasher key. In a similar way, matrix privacy states that the messages sent by the encryptor leak no information about the input matrices. Again, this holds even if the matrix key is reused multiple times.

In general, both properties are guaranteed as long as the hasher key and the matrix key remain secret (similarly to what happens in every symmetric encryption scheme). We now formalise, however, stronger security notions that, if satisfied, ensure the privacy of both parties even if the keys are made public (similarly to all public key encryption schemes).

THE GAME $\mathcal{G}_n^{\mathsf{MP}}(\lambda)$

Initialisation: This procedure is run only once at the beginning of the game.

(a) $b \xleftarrow{\$} \{0,1\}$

(b) Activate $\mathcal{A}$ with $\mathbb{1}^\lambda$ and receive n in unary notation.

(c) $(\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$

(d) Provide hk to the adversary.

Query: This procedure can be run multiple times and at any moment.

(a) Receive an n-column matrix $M \in \mathcal{M}$ from $\mathcal{A}$.

(b) $(E, \phi) \xleftarrow{\$} \mathsf{Matrix}(\mathsf{mk}, M)$

(c) Provide the adversary with E.

Challenge: This procedure can be run only once and at any moment.

(a) Receive two n-column matrices with the same number of rows $M_0, M_1 \in \mathcal{M}$ from $\mathcal{A}$.

(b) $(E, \phi) \xleftarrow{\$} \mathsf{Matrix}(\mathsf{mk}, M_b)$

(c) Provide the adversary with E.

Win: The adversary wins if it ends its execution outputting b.

Fig. 7. The Game for Matrix Privacy

Definition 10 (Strong Hasher Privacy). *Let* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Matrix}, \mathsf{HasherEval}, \mathsf{MatrixEval})$ *be a bilinear HSS scheme over the ring R. We say that the scheme satisfies strong hasher privacy if, for every $n \in \mathbb{N}$ and values $\boldsymbol{x_0}, \boldsymbol{x_1} \in R^n$, the following distributions are computationally indistinguishable*

$$\left\{ (d, \mathsf{hk}, \mathsf{mk}) \;\middle|\; \begin{array}{l} (\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ (d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, \boldsymbol{x_0}) \end{array} \right\} \quad \left\{ (d, \mathsf{hk}, \mathsf{mk}) \;\middle|\; \begin{array}{l} (\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ (d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, \boldsymbol{x_1}) \end{array} \right\}$$

Definition 11 (Strong Matrix Privacy). *Let* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Matrix}, \mathsf{HasherEval}, \mathsf{MatrixEval})$ *be a bilinear HSS scheme over the ring R. We say that the scheme satisfies strong matrix privacy if, or every $n, m \in \mathbb{N}$ and $m \times n$ matrices $M_0, M_1 \in \mathcal{M}$ over R, the following distributions are computationally indistinguishable*

$$\left\{ (E, \mathsf{hk}, \mathsf{mk}) \;\middle|\; \begin{array}{l} (\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ (E, \phi) \xleftarrow{\$} \mathsf{Matrix}(\mathsf{mk}, M_0) \end{array} \right\} \quad \left\{ (E, \mathsf{hk}, \mathsf{mk}) \;\middle|\; \begin{array}{l} (\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ (E, \phi) \xleftarrow{\$} \mathsf{Matrix}(\mathsf{mk}, M_1) \end{array} \right\}$$

If strong hasher privacy and strong matrix privacy hold simultaneously, we say that we are dealing with a public-key bilinear HSS scheme. In these situations, we use pk to denote the concatenation of hk and mk. We modify the syntax of Hash and Matrix by providing them directly with pk instead of just hk and mk.

Definition 12 (Public-Key Bilinear HSS). *A bilinear HSS scheme is public-key if it simultaneously satisfies both strong hasher privacy and strong matrix privacy.*

We finally present a even stronger version of hasher privacy, which we call *transparent hasher privacy*. The latter states that the privacy of the hashed messages holds even if we provide the adversary with the random coins used to run the setup. We will use this property to build our sublinear communication MPC protocols.

Definition 13 (Transparent Hasher Privacy). *Let* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Matrix}, \mathsf{HasherEval}, \mathsf{MatrixEval})$ *be a bilinear HSS scheme over the ring R. Let $L(\lambda, n)$ denote the length of the randomness needed by $\mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$. We say that the scheme satisfies transparent hasher privacy if, for every $n \in \mathbb{N}$ and values $\boldsymbol{x_0}, \boldsymbol{x_1} \in R^n$, the following distributions are computationally indistinguishable*

$$\left\{ (d, \rho) \;\middle|\; \begin{array}{l} \rho \xleftarrow{\$} \{0,1\}^{L(\lambda,n)} \\ (\mathsf{hk}, \mathsf{mk}) \leftarrow \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n; \rho) \\ (d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, \boldsymbol{x_0}) \end{array} \right\} \quad \left\{ (d, \rho) \;\middle|\; \begin{array}{l} \rho \xleftarrow{\$} \{0,1\}^{L(\lambda,n)} \\ (\mathsf{hk}, \mathsf{mk}) \leftarrow \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n; \rho) \\ (d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, \boldsymbol{x_1}) \end{array} \right\}$$

A bilinear HSS scheme guarantees that the size of the digests is sublinear in the size of the hasher input n. This does not prevent however that the size of the keys and matrix encoding scales polynomially in n

(we recall that the length of the hasher input is fed into Setup). If the size of the keys grows sublinearly in n, we say that the scheme satisfies key-compactness. If instead the matrix encoding of any $n \times n$ matrix in the class $\mathcal{M}$ is sublinear in n, we say that the scheme is matrix-compact. Observe that, in order for matrix compactness to be satisfied, the class of $n \times n$ matrices in $\mathcal{M}$ must contain $o(2^n)$ elements. In other words, any bilinear scheme for the class of all matrices $M \in R^{n \times n}$ cannot satisfy matrix compactness.

Definition 14 (Key-Compact Bilinear HSS). *A bilinear HSS scheme over the ring R is key-compact if, for every $\lambda, n \in \mathbb{N}$, the size of the keys $(\mathsf{hk}, \mathsf{mk})$ output by $\mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$ is $\mathsf{poly}(\lambda, \log|R|) \cdot o(n)$.*

Definition 15 (Matrix-Compact Bilinear HSS). *A bilinear HSS scheme over the ring R is matrix-compact if, for every $\lambda, n \in \mathbb{N}$, and matrix $M \in R^{n \times n}$, the size of the encoding E output by $\mathsf{Matrix}(\mathsf{mk}, M)$, where $(\mathsf{hk}, \mathsf{mk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$, is $\mathsf{poly}(\lambda, \log|R|) \cdot o(n)$.*

4 Public-Key Bilinear HSS Constructions

In this section, we show how to build public-key bilinear HSS schemes for all matrices based on DLOG over unknown order groups (such as Paillier or class groups, see Section 4.1) or lattices (see Section 4.2).

4.1 Public-Key Bilinear HSS for all Matrices Based in the NIDLS Framework

We formalise our construction in the NIDLS framework of [ADOS22], which can be instantiated e.g. using Paillier, Goldwasser-Micali where the RSA modulus is product of safe-primes or class groups. We recall that class groups have the advantage of requiring only a transparent setup. We start by formalising the complexity assumptions we need.

New Assumptions in the NIDLS Framework.

The uniformity assumption. We introduce a new computational assumption over the framework, we call it the *uniformity assumption*. The latter essentially says that given the parameters of the NIDLS group, a sample g_0 from $\mathcal{D}$ and the corresponding auxiliary information ρ_0 [9], it is hard to distinguish between a random power of g_0 and another sample g_1 from $\mathcal{D}$ where the corresponding auxiliary information ρ_1 is kept secret.

Definition 16 (The Uniformity Assumption). *We say that the uniformity assumption holds in the NIDLS framework if the following distributions are computationally indistinguishable*

$$\left\{ \mathsf{par}, g_0, \rho_0, g_0^w \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda) \\ (g_0, \rho_0) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par}) \\ w \xleftarrow{\$} [\ell] \end{array} \right\}$$

$$\left\{ \mathsf{par}, g_0, \rho_0, g_1 \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda) \\ (g_0, \rho_0) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par}) \\ (g_1, \rho_1) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par}) \end{array} \right\}$$

The above assumption is useful only when the NIDLS group is not cyclic of prime order. Indeed, if that was not the case, the assumption holds information-theoretically. We observe that the subgroup of squares of the Paillier group $\mathbb{Z}^*_{N^2}$ is cyclic. If N is product of distinct safe-primes, also the subgroup of elements with Jacobi symbol 1 in $\mathbb{Z}^*_N$ is cyclic. So, in these settings the uniformity assumption holds unconditionally.

[9] For instance, ρ_0 can represent the randomness used to produce g_0, if we aim to generate them using common random string or a random oracle.

Class groups, unfortunately, are not cyclic and we could not relate uniformity to any other known assumption. We believe however that uniformity is likely to hold even in this setting. Our claim is supported by the hardness of computing discrete logarithms over class groups. We also highlight that the assumption resembles, to some extent, the DXDH assumption of [ADOS22], which says that given the description of the NIDLS group and two samples g_0, g_1 from $\mathcal{D}$, it is hard to distinguish (g_0^r, g_1^r) from (g_0^r, g_0^s) for random $r, s \xleftarrow{\$} [\ell]$. This implies that it is hard to tell whether a group element belongs to $\langle g_0 \rangle$.

Theorem 3. *If* Gen *outputs a cyclic group* G *of order* N *such that* $\phi(N)/N = 1 - \mathsf{negl}(\lambda)$*,* $\mathcal{D}$ *is the uniform distribution over* G *and* $|\ell - N|/N \leq \mathsf{negl}(\lambda)$*, the uniformity assumption holds information theoretically.*

Proof. We notice that $\mathcal{D}$ outputs a generator of G with overwhelming probability as the number of generators is $\phi(N)$ and $\phi(N)/N$ is negligible. Furthermore, the distribution of w mod N where $w \xleftarrow{\$} [\ell]$ is statistically close to the uniform distribution over $\mathbb{Z}_N$. We conclude that the distribution of g_0^w is statistically close to the uniform distribution over G. □

The n-ary enhanced DDH assumption. We introduce the second assumption needed to build bilinear HSS in the NIDLS framework, we call it *enhanched DDH (EDDH) assumption.* The latter states that given the parameters of the NIDLS group and $n+1$ group elements $g_0, \ldots, g_n$ sampled from $\mathcal{D}$ (along with the corresponding auxiliary information $\rho_0, \ldots, \rho_n$), it is hard to distinguish between $(g_0^w, \ldots, g_n^w)$ for a random w and $(f^{r_0} \cdot g_0^w, \ldots, f^{r_n} \cdot g_n^w)$ for random $r_0, \ldots, r_n \in \mathbb{Z}_q$.

Definition 17 (The n-ary Enhanced DDH Assumption). *We say that the n-ary Enhanced DDH (n-EDDH) assumption holds in the NIDLS framework if the following distributions are computationally indistinguishable*

$$\left\{ \begin{array}{c|l} \mathsf{par} & \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda) \\ g_0, \ldots, g_n & \forall j \in [0..n] : (g_j, \rho_j) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par}) \\ \rho_0, \ldots, \rho_n & w \xleftarrow{\$} [\ell] \\ g_0^w, \ldots, g_n^w & \end{array} \right\}$$

$$\left\{ \begin{array}{c|l} \mathsf{par} & \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda) \\ g_0, \ldots, g_n & \forall j \in [0..n] : (g_j, \rho_j) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par}) \\ \rho_0, \ldots, \rho_n & w \xleftarrow{\$} [\ell] \\ f^{r_0} \cdot g_0^w, \ldots, f^{r_n} \cdot g_n^w & \forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_q \end{array} \right\}$$

The above assumption is related to DDH. For instance, it is easy to see that if the NIDLS group has prime order, and $\mathcal{D}$ outputs a random group element without any auxiliary information, n-EDDH is implied by DDH and the hidden subgroup assumption for any polynomial $n(\lambda)$. Unfortunately, however, all known instantiations of the NIDLS framework do not have prime order (in particular, class groups, the RSA group and the Paillier group are not even cyclic). Moreover, we have often an additional issue because the distribution $\mathcal{D}$ might be different from the uniform distribution and, furthermore, it can be non-explainable (this if for instance the case, as far as we know, over class groups), meaning that, given a group element produced by $\mathcal{D}$, we are not able to simulate the corresponding auxiliary information. This fact prevents the reduction from EDDH to DDH from working as we can no longer substitute e.g. g_j with a random power of g_0 (we would not be able to simulate ρ_j).

Definition 18 (Explainable Sampler). *The distribution $\mathcal{D}$ is explainable if there exists a PPT algorithm* $\mathsf{Explain}$ *such that the following distributions are computationally indistinguishable*

$$\left\{ \mathsf{par}, g, \rho \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \end{array} \right\}$$

$$\left\{ \mathsf{par}, g, \rho' \;\middle|\; \begin{array}{l} \mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ (g, \rho) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par}) \\ \rho' \xleftarrow{\$} \mathsf{Explain}(1\!\!1^\lambda, \mathsf{par}, g) \end{array} \right\}$$

Enhanced DDH over class groups. As we have already mentioned, over class groups, the distribution $\mathcal{D}$ is not known to be explainable (unless we decide to give up on the transparent setup by not revealing the randomness ρ used by the sampling procedure). That prevents the reduction from EDDH to DDH from succeeding. We believe however that n-ary EDDH is likely to hold in this setting for any polynomial $n(\lambda)$. To support our claim, we observe that the points highlighted in [ADOS22] to argue the hardness of DXDH can also be applied to n-ary EDDH.

Enhanced DDH over the Paillier group and the RSA group. Compared to the class group case, analysing the hardness of EDDH over Paillier and Goldwasser-Micali is significantly easier. We show that, for any polynomial $n(\lambda)$, the n-ary EDDH assumption over the Paillier group is implied by DCR and, over Goldwasser-Micali, by the QR assumption.

Theorem 4. *The following are true.*

- *Suppose that* $\mathsf{Gen} = \mathsf{Gen}_{\mathsf{Paillier}}$ *and* $\mathcal{D} = \mathcal{D}_{\mathsf{Paillier}}$*. Then, DCR implies the n-ary EDDH assumption for any polynomial $n(\lambda)$.*
- *Suppose that* $\mathsf{Gen} = \mathsf{Gen}_{\mathsf{GM}}$ *and* $\mathcal{D} = \mathcal{D}_{\mathsf{GM}}$*. Then, QR implies the n-ary EDDH assumption for any polynomial $n(\lambda)$.*

Proof. We prove both points of the theorem simultaneously by relying on the ideas of [BG10, Lemma B.1]. We proceed by considering the following sequence of hybrids.

Hybrid 0**:**

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda)$
2. $\forall j \in [0..n] : (g_j, \rho_j) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $w \xleftarrow{\$} [\ell]$
4. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, g_0^w, \ldots, g_n^w$

Hybrid 1**:**

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda)$
2. $\forall j \in [0..n] : (h_j, \eta_j) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $\forall j \in [0..n] : g_j \leftarrow h_j^q$
4. $\forall j \in [0..n] : \rho_j \leftarrow \eta_j^q$
5. $w \xleftarrow{\$} [\ell]$
6. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, g_0^w, \ldots, g_n^w$

Hybrid $2.i.0$**:**

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda)$
2. $\forall j \in [0..n] : (h_j, \eta_j) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $\forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_q$

4. $\forall j \in [0..n] : g_j \leftarrow h_j^q$
5. $\forall j \in [0..n] : \rho_j \leftarrow \eta_j^q$
6. $w \xleftarrow{\$} [\ell]$
7. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, f^{r_0} \cdot g_0^w, \ldots, f^{r_{i-1}} \cdot g_{i-1}^w, g_i^w, \ldots, g_n^w$

Hybrid $2.i.1$**:** Let $C := 2$ if $\mathsf{Gen} = \mathsf{Gen}_{\mathsf{Paillier}}$, let $C := 1$ otherwise.

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda)$
2. $\forall j \in [0..n] : (h_j, \eta_j) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $\forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_q$
4. $\forall j \in [0..n] \setminus \{i\} : g_j \leftarrow h_j^q$
5. $\forall j \in [0..n] \setminus \{i\} : \rho_j \leftarrow \eta_j^q$
6. $s \xleftarrow{\$} \mathbb{Z}_q$
7. $g_i \leftarrow f^s \cdot h_i^q$
8. $\rho_i \leftarrow f^{s/C} \cdot \eta_i^q$
9. $w \xleftarrow{\$} [\ell]$
10. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, f^{r_0} \cdot g_0^w, \ldots, f^{r_{i-1}} \cdot g_{i-1}^w, g_i^w, \ldots, g_n^w$

Hybrid $2.i.2$**:**

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda)$
2. $\forall j \in [0..n] : (h_j, \eta_j) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $\forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_q$
4. $\forall j \in [0..n] \setminus \{i\} : g_j \leftarrow h_j^q$
5. $\forall j \in [0..n] \setminus \{i\} : \rho_j \leftarrow \eta_j^q$
6. $s \xleftarrow{\$} \mathbb{Z}_q$
7. $g_i \leftarrow f^s \cdot h_i^q$
8. $\rho_i \leftarrow f^{s/C} \cdot \eta_i^q$
9. $w \xleftarrow{\$} [q \cdot \varphi(N)/4]$
10. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, f^{r_0} \cdot g_0^w, \ldots, f^{r_{i-1}} \cdot g_{i-1}^w, g_i^w, \ldots, g_n^w$

Hybrid $2.i.3$**:**

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda)$
2. $\forall j \in [0..n] : (h_j, \eta_j) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $\forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_q$
4. $\forall j \in [0..n] \setminus \{i\} : g_j \leftarrow h_j^q$
5. $\forall j \in [0..n] \setminus \{i\} : \rho_j \leftarrow \eta_j^q$
6. $s \xleftarrow{\$} \mathbb{Z}_q$
7. $g_i \leftarrow f^s \cdot h_i^q$
8. $\rho_i \leftarrow f^{s/C} \cdot \eta_i^q$
9. $w \xleftarrow{\$} [q \cdot \varphi(N)/4]$
10. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, f^{r_0} \cdot g_0^w, \ldots, f^{r_{i-1}} \cdot g_{i-1}^w, f^{r_i} \cdot g_i^w, g_{i+1}^w, \ldots, g_n^w$

Hybrid $2.i.4$**:**

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda)$
2. $\forall j \in [0..n] : (h_j, \eta_j) \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $\forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_q$
4. $\forall j \in [0..n] \setminus \{i\} : g_j \leftarrow h_j^q$
5. $\forall j \in [0..n] \setminus \{i\} : \rho_j \leftarrow \eta_j^q$

6. $s \xleftarrow{\$} \mathbb{Z}_q$
7. $g_i \leftarrow f^s \cdot h_i^q$
8. $\rho_i \leftarrow f^{s/C} \cdot \eta_i^q$
9. $w \xleftarrow{\$} [\ell]$
10. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, f^{r_0} \cdot g_0^w, \ldots, f^{r_{i-1}} \cdot g_{i-1}^w, f^{r_i} \cdot g_i^w, g_{i+1}^w, \ldots, g_n^w$

Hybrid 3:

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux} = N) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda)$
2. $\forall j \in [0..n] : (g_j, \rho_j) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$
3. $\forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_q$
4. $w \xleftarrow{\$} [\ell]$
5. Provide the adversary with $N, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n, f^{r_0} \cdot g_0^w, \ldots, f^{r_n} \cdot g_n^w$

If $\mathsf{Gen} = \mathsf{Gen}_{\mathsf{Paillier}}$, Hybrid 0 and Hybrid 1 are indistinguishable under the DCR assumption and so are also Hybrid 2.i.0 and Hybrid 2.i.1 (for every $i \in [0..n]$), Hybrid 2.i.4 and Hybrid 2.$(i+1)$.0 (for every $i \in [0..n]$), and Hybrid 2.$(n+1)$.0 and Hybrid 3. If instead $\mathsf{Gen} = \mathsf{Gen}_{\mathsf{GM}}$, these hybrids are indistinguishable under the QR assumption.

Hybrid 1 and Hybrid 2.0.0 are identical. Hybrid 2.i.2 and Hybrid 2.i.3 are statistically indistinguishable for every $i \in [0..n]$. This is due to the fact that ℓ is 2^λ times greater than the order of G. Finally, Hybrid 2.i.3 and Hybrid 2.i.4 are perfectly indistinguishable for every $i \in [0..n]$. This is due to the fact that the order of f and the order of h_j^q are coprime for every $j \in [0..n]$. □

The Bilinear HSS Scheme in the NIDLS Framework. We are ready to present our first bilinear HSS scheme. The construction is public-key, supports all $m \times n$ matrices over $\mathbb{Z}_q$, and is secure in the NIDLS framework under the uniformity assumption (see Def. 16), the n-ary EDDH assumption (see Def. 18) and the hidden subgroup assumption. We describe it in Fig. 8. The rationale behind the scheme is explained in Section 1.2. We highlight that the construction does not satisfy key-compactness.

Theorem 5. *If the n-ary EDDH assumption, the uniformity assumption and the hidden subgroup assumption hold in the NIDLS framework [ADOS22], the construction in Fig. 8 is a secure public-key bilinear HSS scheme for the class $\mathcal{M}$ of $m \times n$ matrices over $\mathbb{Z}_q$.*

Moreover, if the NIDLS framework is instantiated using Paillier or Goldwasser-Micali or class groups, the scheme satisfies transparent hasher privacy.

Proof. It is trivial to see that the size of the digest d is independent of n. We start by proving correctness. We observe that, for every $i \in [m]$,

$$\begin{aligned}\left(E_{i,0}^u \cdot \prod_{j=1}^n E_{i,j}^{x_j}\right) \cdot d^{-w_i} &= \left(g_0^{w_i \cdot u} \cdot \prod_{j=1}^n f^{M_{i,j} \cdot x_j} \cdot g_j^{w_i \cdot x_j}\right) \cdot \left(g_0^{-u \cdot w_i} \cdot \prod_{j=1}^n g_j^{-x_j \cdot w_i}\right) \\ &= f^{\sum_{j=1}^n M_{i,j} \cdot x_j}\end{aligned}$$

By the properties of the distributed DLOG procedure, we conclude that, with overwhelming probability, $\boldsymbol{s_0} - \boldsymbol{s_1} = M \cdot \boldsymbol{x}$.

We now proceed by proving the hasher privacy. We do this by relying on a sequence of $n+1$ indistinguishably hybrids, the t-th one of these consisting of the following

Hybrid t.0.

1. $\mathsf{pk} := (\mathsf{par}, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$
2. $u \xleftarrow{\$} [\ell]$
3. $d \leftarrow g_t^u \cdot \cdot g_{t+1}^{x_{t+1}} \cdot \cdots \cdot g_n^{x_n}$

Bilinear HSS for all Matrices based on DDLOG

$\mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$:

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda)$
2. $\forall j \in [0..n] : (g_j, \rho_j) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$
3. Output $\mathsf{pk} := (\mathsf{par}, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n)$.

$\mathsf{Hash}\big(\mathsf{pk} = (\mathsf{par}, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n), \boldsymbol{x}\big)$:

1. $u \xleftarrow{\$} [\ell]$
2. $d \leftarrow g_0^u \cdot \prod_{j=1}^n g_j^{x_j}$
3. $\psi \leftarrow (\mathsf{par}, u, \boldsymbol{x})$
4. Output d and ψ.

$\mathsf{Matrix}\big(\mathsf{pk} = (\mathsf{par}, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n), M\big)$:

1. $\forall i \in [m] : w_i \xleftarrow{\$} [\ell]$
2. $\forall i \in [m] : E_{i,0} \leftarrow g_0^{w_i}$
3. $\forall i \in [m], j \in [n] : E_{i,j} \leftarrow f^{M_{i,j}} \cdot g_j^{w_i}$
4. Output $E := (E_{i,j})_{i,j}$ and $\phi = (\mathsf{par}, w_1, \ldots, w_m, M)$.

$\mathsf{HasherEval}\big(E, \psi = (\mathsf{par}, u, \boldsymbol{x})\big)$:

1. $\forall i \in [m] : s_{0,i} \leftarrow \mathsf{DDLog}(\mathsf{par}, E_{i,0}^u \cdot \prod_{j=1}^n E_{i,j}^{x_j})$
2. Output $\boldsymbol{s_0}$

$\mathsf{MatrixEval}\big(d, \phi = (\mathsf{par}, w_1, \ldots, w_m, M)\big)$:

1. $\forall i \in [m] : s_{1,i} \leftarrow \mathsf{DDLog}(\mathsf{par}, d^{w_i})$
2. Output $\boldsymbol{s_1}$

Fig. 8. Bilinear HSS for all Matrices based on DDLOG

4. Output (pk, d)

Hybrid t.1.

1. $\mathsf{pk} := (\mathsf{par}, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$
2. $(h, \rho') \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$
3. $d \leftarrow h \cdot g_{t+1}^{x_{t+1}} \cdot \cdots \cdot g_n^{x_n}$
4. Output (pk, d)

Observe that under the uniformity assumption, the Hybrid $t.0$ is indistinguishable from Hybrid $t.1$. Furthermore, under the same assumption, Hybrid $t.1$ is indistinguishable from Hybrid $(t+1).0$. We observe that in Hybrid 0.0, the distribution of (pk, d) is as if d was produced by $\mathsf{Hash}(\mathsf{pk}, \boldsymbol{x})$. In Hybrid $(n-1).1$, d is independent of $\boldsymbol{x}$. Hasher privacy immediately follows.

Next, we prove matrix privacy. We do this by relying on the following sequence of indistinguishably hybrids.

Hybrid 0.

1. $\mathsf{pk} := (\mathsf{par}, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$
2. $\forall i \in [m] : w_i \xleftarrow{\$} [\ell]$
3. $\forall i \in [m] : E_{i,0} \leftarrow g_0^{w_i}$
4. $\forall i \in [m], j \in [n] : E_{i,j} \leftarrow f^{M_{i,j}} \cdot g_j^{w_i}$
5. Output (pk, E)

Hybrid 1.

1. $\mathsf{pk} := (\mathsf{par}, g_0, \ldots, g_n, \rho_0, \ldots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$
2. $\forall i \in [m] : w_i \xleftarrow{\$} [\ell]$
3. $\forall i \in [m], j \in [n] : r_{i,j} \xleftarrow{\$} \mathbb{Z}_q$
4. $\forall i \in [m] : E_{i,0} \leftarrow f^{r_{i,0}} \cdot g_0^{w_i}$
5. $\forall i \in [m], j \in [n] : E_{i,j} \leftarrow f^{M_{i,j}+r_{i,j}} \cdot g_j^{w_i}$
6. Output (pk, E)

Observe that Hybrid 0 and Hybrid 1 are indistinguishable under the n-ary EDDH assumption. We conclude the proof by observing that the distribution of (pk, E) in Hybrid 0 is exactly as if E was generated by $\mathsf{Matrix}(\mathsf{pk}, M)$. Furthermore, the distribution of (pk, E) in Hybrid 1 is independent of M as all information is masked by $(r_{i,j})_{i\in[m],j\in[n]}$. □

4.2 Public-Key Bilinear HSS for all Matrices Based on Lattices

The techniques of the construction in Section 4.1 can be adapted to work over lattices, similarly to how the techniques of [BGI16] were used in [BKS19]. In this section, we show how this is done in detail.

Since the resulting scheme can be instantiated under multiple variants of the LWE problem, such as plain LWE or Ring-LWE, we abstract out how the LWE matrix and the noise are sampled. In particular, we rely on a randomised algorithm LWEGen that, on input $\mathbb{1}^\lambda$, $\mathbb{1}^n$, $p \in \mathbb{N}$, produces matrices $A \in \mathbb{Z}_q^{n\times k}$, $B \in \mathbb{Z}_q^{t\times k}$, for some $q, t, k \in \mathbb{N}$, along with descriptions of noise distributions. We require that primal LWE holds with respect to the matrix $(A \parallel B)^\intercal$. Furthermore, we require that dual LWE holds with respect to B. Finally, we ask that the noise produced by the given distributions remains small relative to q/p. Below, we formalise the exact properties we desire.

Definition 19 (LWE sampler). *An LWE sampler is a PPT algorithm* LWEGen *that, on input the security parameter* $\mathbb{1}^\lambda$, $\mathbb{1}^n$ *and* $p \in \mathbb{N}$, *produces a positive integers* q, t, k, *matrices* $A \in \mathbb{Z}_q^{n\times k}, B \in \mathbb{Z}_q^{t\times k}$ *and distributions* ξ, χ.

We require the following properties:

- ***(Primal).*** *For every* $n, p \in \mathbb{N}$, *the following distributions are computationally indistinguishable*

$$\left\{ \begin{pmatrix} q, A, B, \xi, \chi \\ A^\intercal \boldsymbol{w} + \boldsymbol{e} \\ B^\intercal \boldsymbol{w} + \boldsymbol{e'} \end{pmatrix} \middle| \begin{array}{l} (q, t, k, A, B, \xi, \chi) \xleftarrow{\$} \mathsf{LWEGen}(\mathbb{1}^\lambda, \mathbb{1}^n, p) \\ \boldsymbol{w} \xleftarrow{\$} \mathbb{Z}_q^k \\ (\boldsymbol{e}, \boldsymbol{e'}) \xleftarrow{\$} \chi \end{array} \right\}$$

$$\left\{ \begin{pmatrix} q, A, B, \xi, \chi \\ \boldsymbol{v}, \boldsymbol{v'} \end{pmatrix} \middle| \begin{array}{l} (q, t, k, A, B, \xi, \chi) \xleftarrow{\$} \mathsf{LWEGen}(\mathbb{1}^\lambda, \mathbb{1}^n, p) \\ \boldsymbol{v} \xleftarrow{\$} \mathbb{Z}_q^n \\ \boldsymbol{v'} \xleftarrow{\$} \mathbb{Z}_q^t \end{array} \right\}$$

- ***(Dual).*** *For every* $n, p \in \mathbb{N}$, *the following distributions are computationally indistinguishable*

$$\left\{ \begin{pmatrix} q, A, B, \xi, \chi \\ B \cdot \boldsymbol{u} \end{pmatrix} \middle| \begin{array}{l} (q, t, k, A, B, \xi, \chi) \xleftarrow{\$} \mathsf{LWEGen}(\mathbb{1}^\lambda, \mathbb{1}^n, p) \\ \boldsymbol{u} \xleftarrow{\$} \xi \end{array} \right\}$$

$$\left\{ \begin{pmatrix} q, A, B, \xi, \chi \\ \boldsymbol{v} \end{pmatrix} \middle| \begin{array}{l} (q, t, k, A, B, \xi, \chi) \xleftarrow{\$} \mathsf{LWEGen}(\mathbb{1}^\lambda, \mathbb{1}^n, p) \\ \boldsymbol{v} \xleftarrow{\$} \mathbb{Z}_q^k \end{array} \right\}$$

- ***(Small noise).*** *There exists a function* $T(\lambda)$ *such that, for every* $\boldsymbol{x} \in \mathbb{Z}_p^n$

$$\Pr\left[|\boldsymbol{x}^\intercal \cdot \boldsymbol{e} + \boldsymbol{u}^\intercal \cdot \boldsymbol{e'}| > T(\lambda) \;\middle|\; \begin{array}{l} (q, t, k, A, B, \xi, \chi) \xleftarrow{\$} \mathsf{LWEGen}(\mathbb{1}^\lambda, \mathbb{1}^n, p) \\ (\boldsymbol{e}, \boldsymbol{e'}) \xleftarrow{\$} \chi, \boldsymbol{u} \xleftarrow{\$} \xi \end{array} \right] \leq \mathsf{negl}(\lambda)$$

and $p \cdot T(\lambda)/q \leq \mathsf{negl}(\lambda)$.

Bilinear HSS for all Matrices based on Lattices

$\mathsf{Setup}(1\!\!1^\lambda, 1\!\!1^n)$:

1. $(q, t, k, A, B, \xi, \chi) \xleftarrow{\$} \mathsf{LWEGen}(1\!\!1^\lambda, 1\!\!1^n, p)$
2. Output $\mathsf{pk} = (A, B)$.

$\mathsf{Hash}\big(\mathsf{pk} = (A, B), \boldsymbol{x}\big)$:

1. $\boldsymbol{u} \xleftarrow{\$} \xi$
2. $d \leftarrow A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u}$
3. $\psi \leftarrow (A, B, \boldsymbol{u}, \boldsymbol{x})$
4. Output d and ψ.

$\mathsf{Matrix}\big(\mathsf{pk} = (A, B), M\big)$:

1. $\forall i \in [m] : \boldsymbol{w_i} \xleftarrow{\$} \mathbb{Z}_q^k$
2. $\forall i \in [m] : (\boldsymbol{e_i}, \boldsymbol{e'_i}) \xleftarrow{\$} \chi$
3. $\forall i \in [m] : E_i \leftarrow A^\intercal \cdot \boldsymbol{w_i} + \boldsymbol{e_i} + \lceil q/p \rfloor \cdot M_i^\intercal$
4. $\forall i \in [m] : E'_i \leftarrow B^\intercal \cdot \boldsymbol{w_i} + \boldsymbol{e'_i}$
5. Output $E := (E_i, E'_i)_{i \in [m]}$ and $\phi = (A, B, \boldsymbol{w_1}, \ldots, \boldsymbol{w_m}, M)$.

$\mathsf{HasherEval}\big(E, \psi = (A, B, \boldsymbol{u}, \boldsymbol{x})\big)$:

1. $\forall i \in [m] : s_{0,i} \leftarrow \lceil \boldsymbol{x}^\intercal \cdot E_i + \boldsymbol{u}^\intercal \cdot E'_i \rfloor_{\lceil q/p \rfloor}$
2. Output $\boldsymbol{s_0}$

$\mathsf{MatrixEval}\big(d, \phi = (A, B, \boldsymbol{w_1}, \ldots, \boldsymbol{w_m}, M)\big)$:

1. $\forall i \in [m] : s_{1,i} \leftarrow \lceil d^\intercal \cdot \boldsymbol{w_i} \rfloor_{\lceil q/p \rfloor}$
2. Output $\boldsymbol{s_1}$

Fig. 9. Bilinear HSS for all Matrices based on Lattices

We are ready to present our second public-key bilinear HSS scheme for all matrices. Its description can be found in Fig. 9, the rationale behind the construction is explained in Section 1.2. Below, we prove security under the assumption that LWEGen satisfies Def. 19. We highlight that even this construction does not satisfy key-compactness.

Theorem 6. *If* LWEGen *is a LWE sampler, the construction in Fig. 9 is a secure public-key bilinear HSS scheme for the class* $\mathcal{M}$ *of all matrices over* $\mathbb{Z}_p$.

Moreover, if LWEGen *is instantiated using plain LWE (or Ring-LWE) with superpolynomial modulus-to-noise ratio, the scheme satisfies transparent hasher privacy.*

Proof. We start by proving correctness. We observe that, for every $i \in [m]$,

$$\begin{aligned}
&(\boldsymbol{x}^\intercal \cdot E_i + \boldsymbol{u}^\intercal \cdot E'_i) - d^\intercal \cdot \boldsymbol{w_i} = \\
&= (\boldsymbol{x}^\intercal \cdot A^\intercal \cdot \boldsymbol{w_i} + \boldsymbol{x}^\intercal \cdot \boldsymbol{e_i} + \boldsymbol{u}^\intercal \cdot B^\intercal \cdot \boldsymbol{w_i} + \boldsymbol{u}^\intercal \cdot \boldsymbol{e'_i} + \lceil q/p \rfloor \cdot M_i \cdot \boldsymbol{x}) \\
&- (\boldsymbol{x}^\intercal \cdot A^\intercal \cdot \boldsymbol{w_i} + \boldsymbol{u}^\intercal \cdot B^\intercal \cdot \boldsymbol{w_i}) \\
&= \lceil q/p \rfloor \cdot M_i \cdot \boldsymbol{x} + \boldsymbol{x}^\intercal \cdot \boldsymbol{e_i} + \boldsymbol{u}^\intercal \cdot \boldsymbol{e'_i}.
\end{aligned}$$

We also observe that $\boldsymbol{w_i}$ is random, d is indistinguishable from random, and k is $\omega(\log \lambda)$[10], so $z_{1,i} := d^\intercal \cdot \boldsymbol{w_i}$ is also indistinguishable from random. Thanks to this fact, along with the small noise property of LWEGen, we conclude that, with overwhelming probability, $\boldsymbol{s_0} + \boldsymbol{s_1} = M \cdot \boldsymbol{x}$.

[10] We need this last property because q may not be prime. If $k = \omega(\log \lambda)$, we are sure that with overwhelming probability one of the entries of $\boldsymbol{w_i}$ is in $\mathbb{Z}_q^*$.

Indeed, suppose that $z_{1,i} = s_{1,i} \cdot \lceil q/p \rfloor + s'_{1,i}$ where $|s'_{1,i}| \leq \frac{1}{2}\lceil q/p \rfloor$. We know that

$$z_{0,i} := \boldsymbol{x}^\intercal \cdot E_i + \boldsymbol{u}^\intercal \cdot E'_i = (s_{1,i} + M_i \cdot \boldsymbol{x}) \cdot \lceil q/p \rfloor + s'_{1,i} + \widetilde{e}_i$$

where $\widetilde{e}_i$ is some error term of magnitude ad most $T(\lambda)$. We observe that $|s_{1,i} - M_i \cdot \boldsymbol{x}| \leq n \cdot p^2/2$. We rewrite $s_{1,i} - M_i \cdot \boldsymbol{x}$ as $s_{0,i} + p \cdot \rho_{0,i}$ where $s_{0,i} \in \mathbb{Z}_p$, and we observe that

$$z_{0,i} = s_{0,i} \cdot \lceil q/p \rfloor + \rho_{0,i} \cdot (p \cdot \lceil q/p \rfloor) + s'_{1,i} + \widetilde{e}_i.$$

Furthermore, since $|p \cdot \lceil q/p \rfloor| \leq p/2$, with overwhelming probability, $|\rho_{0,i} \cdot (p \cdot \lceil q/p \rfloor) + \widetilde{e}_i| \leq T(\lambda) + n \cdot p^2$. Correctness is preserved as long as

$$|\rho_{0,i} \cdot (p \cdot \lceil q/p \rfloor) + s'_{1,i} + \widetilde{e}_i| \leq \frac{1}{2}\lceil q/p \rfloor.$$

When $z_{1,1}$ is at least $(T(\lambda) + p^2)$-away from $(\ell + 1/2) \cdot \lceil q/p \rfloor$ for every $\ell \in \mathbb{Z}_p$, we are sure that the bad event will not occur. The probability of the bad even is therefore bounded from above by $p \cdot (T + n \cdot p^2)/q$, which is negligible.

We can easily prove hasher privacy by the dual security of LWEGen. Indeed, $B \cdot \boldsymbol{u}$ is indistinguishable from random. Such value masks all information about $\boldsymbol{x}$ hidden in d.

In a similar way, under the primal security of LWEGen, we can show that E leaks no information about M. This concludes the proof. □

5 Succinct Half-Chosen Vector OLE

In this section and in Section 6, we study bilinear HSS schemes for the class $\mathcal{K}$ of matrices that are multiples of the identity. Formally, denoting the ring by R,

$$\mathcal{K} := \{k \cdot \mathsf{id}_m | m \in \mathbb{N}, k \in R\}$$

This class is particularly interesting because any bilinear HSS scheme for $\mathcal{K}$ immediately gives a "half-chosen" non-interactive Vector-OLE (VOLE) protocol: in a VOLE, one party, called the *Sender*, inputs two vectors $\boldsymbol{x}, \boldsymbol{y}$, whereas the other party, called the *Receiver*, inputs a scalar k. At the end of the protocol, the Receiver obtains $\boldsymbol{z} := k \cdot \boldsymbol{x} + \boldsymbol{y}$ without learning anything else (the Sender learns no information at all). Observe that the pair $(\boldsymbol{z}, \boldsymbol{y})$ consists of a secret-sharing of $k \cdot \boldsymbol{x}$. In a fully random VOLE protocol, on the other hand, the parties have no inputs. At the end of the protocol, the Sender will receive random vectors $\boldsymbol{x}, \boldsymbol{y}$, whereas the Receiver will obtain a random constant k along with $\boldsymbol{z} = k \cdot \boldsymbol{x} + \boldsymbol{y}$ (the parties will learn no other information). Choosing $\boldsymbol{x}$ later requires sending a full correction vector $\boldsymbol{x}' - \boldsymbol{x}$. Our VOLE protocol will be a middle ground between these two notions: the Sender will only input $\boldsymbol{x}$, and the Receiver will only input the constant k. The output will be a random secret-sharing of $k \cdot \boldsymbol{x}$.

We probably do not need to explain how important VOLE protocols are in cryptography. What we want to argue is, however, that, in many applications, we do not need a true VOLE protocol – a half-chosen VOLE is sufficient. The surprising fact is that, while it can be proven that VOLE protocols require $\Omega(n)$ communication[11] (n denotes the length of the vectors $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$), half-chosen VOLE protocols can achieve sublinear communication in n. This fact was not known before this work, in this section, we show how bilinear HSS allows us to achieve this.

Then, in Section 6, we focus our energy in constructing bilinear HSS schemes for $\mathcal{K}$ where the communication complexity of the encryptor is $o(n^2)$. Due to an entropy matter, this is of course impossible to achieve in bilinear HSS scheme for all matrices.

[11] If that wasn't the case, we could send a length-n message m with $o(n)$ communication by just inputting $\boldsymbol{x} = \boldsymbol{0}$ and $\boldsymbol{y} = m$ in the VOLE protocol.

5.1 Succinct Half-Chosen VOLE and Key-Compact, Matrix-Compact Bilinear HSS

We consider the half-chosen non-interactive Vector OLE problem (VOLE): Alice holds a vector $\boldsymbol{x} \in \mathbb{Z}_p^n$ whereas Bob holds a constant $k \in \mathbb{Z}_p$. We would like to non-interactively obtain an additive secret sharing of $k \cdot \boldsymbol{x}$. We observe that multiplying by the constant k is equivalent to multiplying by the matrix $k \cdot \mathsf{id}_n$. Now, if we would naively use the bilinear HSS schemes presented in Section 4 to perform such operation, the communication of Alice would be independent of n, however, the communication of Bob would be $\Omega(n^2)$. Furthermore, the size of the public key would be $O(n)$. Is it still possible to use our results to build a half-chosen non-interactive VOLE protocol with $o(n)$ total communication?

To answer this question, we rely on the fact that the messages of bilinear HSS schemes can be reused: if the ciphertext sent by Bob (encoding a matrix M) were to be reused with a new message from Alice, let's say a digest of $\boldsymbol{x'}$, the parties would still obtain a secret-sharing of $M \cdot \boldsymbol{x'}$. To achieve sublinear communication in the VOLE protocol, we can therefore apply the following trick: Bob will send an encoding of the matrix $k \cdot \mathsf{id}_{n^{1/3}}$, Alice will instead split its input into $n^{2/3}$ chunks of $n^{1/3}$ elements and will send a digest of each segment. By running the bilinear HSS scheme $n^{2/3}$ times, once for each digest sent by Alice, the parties will therefore obtain a secret-sharing of $k \cdot \boldsymbol{x}$. Surprisingly, the total communication is now $O(n^{2/3})$, equally distributed between Alice and Bob. The size of the public key has also decreased to $O(n^{1/3})$. In other words, our scheme would satisfy both key-compactness and matrix-compactness.

We summarise our idea in the following theorem.

Theorem 7. *Let* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Matrix}, \mathsf{HasherEval}, \mathsf{MatrixEval})$ *be a bilinear HSS scheme over* R *for the matrix class* $\mathcal{M} \supseteq \mathcal{K}$*. Suppose that the size of the HSS digest is upper bounded by* $f(n) \cdot \mathsf{poly}(\lambda, \log|R|)$ *and the encoding of an* $n \times n$ *matrix is upper bounded by* $g(n) \cdot \mathsf{poly}(\lambda, \log|R|)$*. Assume also that the size of the keys is upper bounded by* $\kappa(n) \cdot \mathsf{poly}(\lambda, \log|R|)$*. Then, there exists a bilinear HSS scheme for* $\mathcal{K}$ *with* $g(t) \cdot \mathsf{poly}(\lambda, \log|R|)$ *communication per party and* $\kappa(t) \cdot \mathsf{poly}(\lambda, \log|R|)$ *key size, where* t *is such that* $n = t \cdot g(t)/f(t)$*. Moreover, this transformation preserves the properties of strong hasher privacy, strong matrix privacy, and transparent hasher privacy.*

The following corollary summarises what we know so far about succinct non-interactive half-chosen VOLE with semi-honest security.

Corollary 1. *The following hold:*

- *Under DCR over the Paillier group* $\mathbb{Z}_{N^2}^*$*, there exists a half-chosen, semi-honest, non-interactive VOLE protocol over* $\mathbb{Z}_N$ *with* $O(n^{2/3})$ *communication per party.*
- *Under QR over the the RSA group* $\mathbb{Z}_N^*$ *where* N *is the product of large safe-primes, there exists exists a half-chosen, semi-honest, non-interactive VOLE protocol over* $\mathbb{Z}_2$ *with* $O(n^{2/3})$ *communication per party.*
- *For any prime* $q = \Omega(2^\lambda)$*, under the* $n^{1/3}$*-ary EDDH assumption and the uniformity assumption over class groups, there exists a half-chosen, semi-honest, non-interactive VOLE protocol over* $\mathbb{Z}_q$ *with* $O(n^{2/3})$ *communication.*
- *For any integer* p*, under LWE with superpolynomial modulus-to-noise ratio, there exists a half-chosen, semi-honest, non-interactive VOLE protocol over* $\mathbb{Z}_p$ *with* $O(n^{2/3})$ *communication per party.*

6 Bilinear HSS Constructions for Multiples of the Identity

In this section, we show how to build a bilinear HSS schemes for $\mathcal{K}$ with strong hasher privacy where the communication complexity of the encryptor is $O(n)$ instead of $\Theta(n^2)$.

6.1 Bilinear HSS for Multiples of the Identity from Power DDH

Our first scheme satisfies strong hasher privacy but not strong matrix privacy. The construction is presented in Fig. 10 and is based on the Power-DDH assumption over the NIDLS framework. The rationale behind the construction was explained in Section 1.2. We highlight that, even if we instantiate the framework with class groups, the setup is not transparent. Notice also that the size of the keys is $O(n)$.

Bilinear HSS for Multiples of the Identity from Power DDH

$\mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$:

1. $\mathsf{par} := (G, F, H, f, q, \ell, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda)$
2. $(g, \rho) \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$
3. $\alpha \xleftarrow{\$} [\ell]$
4. $\forall j \in [0..n] \setminus \{n+1\} : g_j \leftarrow g^{\alpha^j}$
5. $\mathsf{hk} \leftarrow (\mathsf{par}, \rho, g_0, \ldots, g_n)$
6. $\mathsf{mk} \leftarrow (\mathsf{par}, \rho, g, \alpha)$
7. Output hk and mk.

$\mathsf{Hash}\big(\mathsf{hk} = (\mathsf{par}, \rho, g_0, \ldots, g_n), \boldsymbol{x}\big)$:

1. $u \xleftarrow{\$} [\ell]$
2. $d \leftarrow g_0^u \cdot \prod_{j=1}^n g_j^{x_j}$
3. $\psi \leftarrow (u, \boldsymbol{x})$
4. Output d and ψ.

$\mathsf{Matrix}\big(\mathsf{mk} = (\mathsf{par}, \rho, g, \alpha), M = k \cdot \mathsf{id}_n\big)$:

1. $r \xleftarrow{\$} [\ell]$
2. $\forall j \in [0..2n-1] \setminus \{n\} : E_j \leftarrow g^{r \cdot \alpha^j}$
3. $E_n \leftarrow f^k \cdot g^{r \cdot \alpha^n}$
4. Output $E = (E_0, \ldots, E_{2n-1})$ and $\phi := r$.

$\mathsf{HasherEval}\big(\mathsf{hk} = (\mathsf{par}, \rho, g_0, \ldots, g_n), E = (E_0, \ldots, E_{2n-1}), \psi = (u, \boldsymbol{x})\big)$:

1. $\forall i \in [n] : s_{0,i} \leftarrow \mathsf{DDLog}\left(\mathsf{par}, E_{n-i}^u \cdot \prod_{j=1}^n E_{n-i+j}^{x_j}\right)$
2. Output $\boldsymbol{s_0}$

$\mathsf{MatrixEval}\big(\mathsf{mk} = (\mathsf{par}, \rho, g, \alpha), d, \phi = r\big)$:

1. $\forall i \in [n] : s_{1,i} \leftarrow \mathsf{DDLog}\left(\mathsf{par}, d^{r \cdot \alpha^{n-i}}\right)$
2. Output $\boldsymbol{s_1}$

Fig. 10. Bilinear HSS for Multiples of the Identity from Power DDH

Theorem 8. *If the $2n$-ary Power-DDH assumption, the uniformity assumption and the $2n$-ary EDDH assumption hold in the NIDLS framework, the construction in Fig. 10 is a secure bilinear HSS scheme over $\mathbb{Z}_q$ satisfying transparent hasher privacy.*

Proof. We start by proving correctness. We observe that, for every $i \in [n]$,

$$\left(E^u_{n-i} \cdot \prod_{j=1}^{n} E^{x_j}_{n-i+j}\right) \cdot d^{-r\cdot\alpha^{n-i}} =$$

$$= \left(g^{u\cdot r\cdot\alpha^{n-i}} \cdot f^{k\cdot x_i} \cdot \prod_{j=1}^{n} g^{x_j\cdot r\cdot\alpha^{n-i+j}}\right) \cdot \left(g^{-u\cdot r\cdot\alpha^{n-i}} \cdot \prod_{j=1}^{n} g^{-x_j\cdot r\cdot\alpha^{n-i+j}}\right) = f^{k\cdot x_i}.$$

We conclude that, with overwhelming probability,

$$\mathsf{DDLog}\left(\mathsf{par}, E^u_{n-i} \cdot \prod_{j=1}^{n} E^{x_j}_{n-i+j}\right) - \mathsf{DDLog}\left(\mathsf{par}, d^{-r\cdot\alpha^{n-i}}\right) = k \cdot x_i.$$

Proving transparent hasher privacy is straightforward: all information about $\boldsymbol{x}$ in d is masked by $g_0^u = g^u$. The distribution of this element is statistically indistinguishable from the uniform distribution over $\langle g\rangle$.

Proving matrix privacy is almost as simple. We observe that, under the uniform assumption, under Power-DDH, $g^{r\cdot\alpha}, \ldots, g^{r\cdot\alpha^{2n-1}}$ are indistinguishable from elements $g^{r\cdot\alpha_1}, \ldots, g^{r\cdot\alpha_{2n-1}}$ for $\alpha_1, \ldots, \alpha_{2n-1} \xleftarrow{\$} [\ell]$. Then, under the uniformity assumption, we switch to the hybrid in which $g^{\alpha_1}, \ldots, g^{\alpha_{2n-1}}$ are substituted with random samples $g_1, \ldots, g_{2n-1}$ from $\mathcal{D}(1\!\!1^\lambda, \mathsf{par})$. Notice that at this point, f^k is masked by $g^r, g_1^r, \ldots, g_{2n-1}^r$. We conclude by arguing that, under the $2n$-ary EDDH assumption, no information about k is leaked. This ends the proof. □

By applying Theorem 7 to build succinct non-interactive VOLE protocols, we obtain the following corollary.

Corollary 2. *The following hold:*

- *Under DCR+Power-DDH over the Paillier group $\mathbb{Z}^*_{N^2}$, there exists a half-chosen, semi-honest, non-interactive VOLE protocol over $\mathbb{Z}_N$ where the communication complexity is $O(\sqrt{n})$.*
- *Under QR+Power-DDH over the RSA group $\mathbb{Z}^*_N$ where N is the product of large random safe-primes, there exists a half-chosen, semi-honest, non-interactive VOLE protocol over $\mathbb{Z}_2$ where the communication complexity is $O(\sqrt{n})$.*
- *For any prime $q = \Omega(2^\lambda)$, under the uniformity assumption, the EDDH assumption and Power-DDH over class groups, there exists a half-chosen, semi-honest, non-interactive VOLE protocol over $\mathbb{Z}_q$ where the communication complexity is $O(\sqrt{n})$.*

6.2 Public-Key Bilinear HSS for Multiples of the Identity from Power Ring-LWE

In this section, we adapt the ideas based on Power-DDH (see Section 6.1) to the lattice setting. In this way, we obtain a bilinear HSS scheme for $\mathcal{K}$ where the communication of the encryptor is $O(n)$. Interesting, differently from the DLOG-based construction in Section 6.1, the scheme we obtain is public-key.

The Power Ring-LWE Assumption. The bilinear HSS scheme we are going to present is based on a new assumption we called *Power Ring-LWE*. We recall that the Ring-LWE assumption with ring R and noise distribution χ states that the following are indistinguishable

$$\left\{ a_1, \ldots, a_m \;\; u_1, \ldots, u_m \;\middle|\; \begin{array}{l} \forall i \in [m] : a_i \xleftarrow{\$} R \\ w \xleftarrow{\$} R \\ \forall i \in [m] : e_i \xleftarrow{\$} \chi \\ \forall i \in [m] : u_i \leftarrow a_i \cdot w + e_i \end{array} \right\} \quad \left\{ a_1, \ldots, a_m \;\; u_1, \ldots, u_m \;\middle|\; \begin{array}{l} \forall i \in [m] : a_i \xleftarrow{\$} R \\ \forall i \in [m] : u_i \xleftarrow{\$} R \end{array} \right\}$$

In the Power Ring-LWE assumption, we require that indistinguishability holds even if $a_1, \ldots, a_m$ are no longer random, but the consists of the first m powers of a random ring element $a \xleftarrow{\$} R$.

Before formalising the definition of Power Ring-LWE, we recall the definition of norm over rings.

Definition 20. *Let $I = \langle f(X) \rangle$ be an ideal of $\mathbb{Z}[X]$ where f is a monic polynomial of degree K. Let R be the ring $\mathbb{Z}[X]/I$. For any $r \in R$, we define the infinity norm $\|r\|_\infty$ as $\max_{i \in [1..M]} |r_i|$ where $(r_1, \ldots, r_M)$ is the only vector in $\mathbb{Z}^M$ such that $r = r_1 + r_2 \cdot X + \cdots + r_M \cdot X^{M-1} + I$.*

Definition 21 (Power Ring LWE). *For every $\lambda \in \mathbb{N}$, let $I_\lambda \langle f_\lambda(X) \rangle$ be an ideal of $\mathbb{Z}[X]$ where f_λ is a monic polynomial. Let $R(\lambda) := \mathbb{Z}[X]/I_\lambda$. Let $R_q(\lambda) = \mathbb{Z}_q[X]/I_\lambda$. Consider positive integers $q(\lambda), m(\lambda) \in \mathbb{N}$ and distributions $\Gamma(\mathbb{1}^\lambda)$ and $\chi(\mathbb{1}^\lambda)$ over $\mathcal{R}_q(\lambda)$. We say that Power Ring-LWE holds with respect to (q, m, R, Γ, χ) if the following distributions are computationally indistinguishable*

$$\left\{ a, v_0, \ldots, v_m \;\middle|\; \begin{array}{l} a \xleftarrow{\$} \Gamma(\mathbb{1}^\lambda) \\ w \xleftarrow{\$} R_q(\lambda) \\ \forall i \in [0..m] : e_i \xleftarrow{\$} \chi(\mathbb{1}^\lambda) \\ \forall i \in [0..m] : v_i \leftarrow a^i \cdot w + e_i \end{array} \right\}$$

$$\left\{ a, v_0, \ldots, v_m \;\middle|\; \begin{array}{l} a \xleftarrow{\$} \Gamma(\mathbb{1}^\lambda) \\ \forall i \in [0..m] : v_i \xleftarrow{\$} R_q(\lambda) \end{array} \right\}$$

The Construction Based on Power Ring-LWE We are now ready to present our bilinear HSS scheme for $\mathcal{K}$ based on the Power Ring-LWE assumption. The scheme is formalised in Fig. 11. The rationale behind the construction was described in Section 1.2. We highlight that the scheme is public-key but does not satisfy key-compactness.

Theorem 9. *Let p, q be security-parameter-dependent positive integers, let I be an ideal of $\mathbb{Z}[X]$ and define $R := \mathbb{Z}[X]/I$, $R_q := \mathbb{Z}_q[X]/I$ and $R_p := \mathbb{Z}_p[X]/I$. Let Γ be a distribution over R_q. Consider a symmetric noise distribution χ [12] such that, there exists a function $T(\lambda)$ such that $p \cdot T(\lambda)/q \leq \mathsf{negl}(\lambda)$ and, for every $\boldsymbol{x} \in R_p^n$,*

$$\Pr\left[\left\| \sum_{j=0}^{1} u_j \cdot e_j + \sum_{j=1}^{n} x_j \cdot e_{j+1} \right\|_\infty > T(\lambda) \;\middle|\; \begin{array}{l} \forall j \in \{0,1\} : u_j \xleftarrow{\$} \chi(\mathbb{1}^\lambda) \\ \forall j \in [0..n+1] : e_j \xleftarrow{\$} \chi(\mathbb{1}^\lambda) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

Under Power Ring-LWE with respect to (q, n, R, Γ, χ), the construction in Fig. 11 is a secure public-key bilinear HSS scheme over $\mathbb{Z}_p$ satisfying transparent hasher privacy.

Proof. We start by proving correctness. We observe that, for every $i \in [n]$,

$$\begin{aligned} &\left(u_0 \cdot E_{n-i} + u_1 \cdot E_{n-i+1} + \sum_{j=1}^{n} x_j \cdot E_{n-i+j+1} \right) - \left(d \cdot a^{n-i} \cdot w \right) = \\ &= u_0 \cdot a^{n-i} \cdot w + u_0 \cdot e_{n-i} + u_1 \cdot a^{n-i+1} \cdot w + u_1 \cdot e_{n-i+1} \\ &\qquad + \sum_{j=1}^{n} \left(x_j \cdot a^{n-i+j+1} \cdot w + x_j \cdot e_{n-i+j+1} \right) + \lceil q/p \rfloor \cdot k \cdot x_i \\ &\qquad\qquad - \left(u_0 \cdot a^{n-i} \cdot w + u_1 \cdot a^{n-i+1} \cdot w + \sum_{j=1}^{n} x_j \cdot a^{n-i+j+1} \cdot w \right) = \\ &= \lceil q/p \rfloor \cdot k \cdot x_i + \sum_{j=0}^{1} u_j \cdot e_{n-i+j} + \sum_{j=1}^{n} x_j \cdot e_{n-i+j+1}. \end{aligned}$$

[12] A distribution χ is symmetric if χ and $-\chi$ coincide.

PUBLIC-KEY BILINEAR HSS FOR MULTIPLES OF THE IDENTITY FROM POWER RING-LWE

$\mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$:

1. $a \xleftarrow{\$} \Gamma(\mathbb{1}^\lambda)$
2. Output $\mathsf{hk} := a$ and $\mathsf{mk} := a$.

$\mathsf{Hash}\big(\mathsf{hk} = a, \boldsymbol{x}\big)$:

1. $u_0, u_1 \xleftarrow{\$} \chi(\mathbb{1}^\lambda)$
2. $d \leftarrow u_0 + a \cdot u_1 + \sum_{j=1}^{n} x_j \cdot a^{j+1}$
3. $\psi \leftarrow (d, \boldsymbol{u}, \boldsymbol{x})$
4. Output d and ψ.

$\mathsf{Matrix}\big(\mathsf{mk} = a, M = k \cdot \mathsf{id}_n\big)$:

1. $w \xleftarrow{\$} R_q$
2. $\forall j \in [0..2n] : e_j \xleftarrow{\$} \chi(\mathbb{1}^\lambda)$
3. $\forall j \in [0..2n] \setminus \{n+1\} : E_j \leftarrow a^j \cdot w + e_j$
4. $E_{n+1} \leftarrow a^{n+1} \cdot w + e_{n+1} + \lceil q/p \rfloor \cdot k$
5. Output $E := (E_0, \ldots, E_{2n})$ and $\phi := w$.

$\mathsf{HasherEval}\big(\mathsf{hk} = a, E = (E_0, \ldots, E_{2n}), \psi = (d, \boldsymbol{u}, \boldsymbol{x})\big)$:

1. $\forall i \in [n] : s_{0,i} \leftarrow \left\lceil \sum_{j=0}^{1} u_j \cdot E_{n-i+j} + \sum_{j=1}^{n} x_j \cdot E_{n-i+j+1} \right\rfloor_{\lceil q/p \rfloor}$
2. Output $\boldsymbol{s_0}$

$\mathsf{MatrixEval}\big(\mathsf{hk} = a, d, \phi = w\big)$:

1. $\forall i \in [n] : s_{1,i} \leftarrow \left\lceil d \cdot a^{n-i} \cdot w \right\rfloor_{\lceil q/p \rfloor}$
2. Output $\boldsymbol{s_1}$

Fig. 11. Public-Key Bilinear HSS for Multiples of the Identity from Power Ring-LWE

In order to complete the proof of correctness, we need to show that the probability that

$$z_{0,i} := u_0 \cdot E_{n-i} + u_1 \cdot E_{n-i+1} + \sum_{j=1}^{n} x_j \cdot E_{n-i+j+1}$$

is at most $(T(\lambda) + p^2)$-away from $(\ell + 1/2) \cdot \lceil q/p \rfloor$ for some $\ell \in \mathbb{Z}_p$ is negligible. Indeed, when that event does not occur, the correctness of the scheme is guaranteed: suppose that $z_{0,i} = s_{0,i} \cdot \lceil q/p \rfloor + s'_{0,i}$ where $\|s'_{0,i}\|_\infty \leq \frac{1}{2}\lceil q/p \rfloor$. We know that $z_{1,i} := d \cdot a^{n-i} \cdot w = (s_{0,i} - k \cdot x_i) \cdot \lceil q/p \rfloor + s'_{0,i} + \widetilde{e}_i$ where $\widetilde{e}_i$ is some error term of magnitude ad most $T(\lambda)$. We observe that $\|s_{0,i} - k \cdot x_i\|_\infty \leq p^2/2$. We rewrite $s_{0,i} - k \cdot x_i$ as $s_{1,i} + p \cdot \rho_{1,i}$ where $s_{1,i} \in R_p$, and we observe that

$$z_{1,i} = s_{1,i} \cdot \lceil q/p \rfloor + \rho_{1,i} \cdot (p \cdot \lceil q/p \rfloor) + s'_{0,i} + \widetilde{e}_i.$$

Furthermore, since $|p \cdot \lceil q/p \rfloor| \leq p/2$, with overwhelming probability, $\|\rho_{1,i} \cdot (p \cdot \lceil q/p \rfloor) + \widetilde{e}_i\|_\infty \leq T(\lambda) + p^2$. Correctness is preserved as long as

$$\|\rho_{1,i} \cdot (p \cdot \lceil q/p \rfloor) + s'_{0,i} + \widetilde{e}_i\|_\infty \leq \frac{1}{2}\lceil q/p \rfloor.$$

When $z_{0,1}$ is at least $(T(\lambda)+p^2)$-away from $(\ell+1/2)\cdot\lceil q/p \rfloor$ for every $\ell \in \mathbb{Z}_p$, we are sure that the bad event will not occur.

Now, we cannot directly argue that $\boldsymbol{z_0} = (z_{0,1}, \ldots, z_{0,n})$ looks random, but we can argue that, if we sample $\widetilde{\boldsymbol{u}} \xleftarrow{\$} \chi^n$, the vector $\boldsymbol{u} + \boldsymbol{z_0}$ looks random under Power Ring-LWE. Indeed, as we are going to show below, $E_0, \ldots, E_{2n}$ are indistinguishable from random. We notice that one of the terms of $z_{0,i} + \widetilde{u}_i$ is $\widetilde{u}_i + u_0 \cdot E_{n-i}$. Under Ring-LWE with lattice distribution given by Γ (which is implied by Power Ring-LWE), such elements are indistinguishable from random. This fact, along the property that $p \cdot T/q$ is negligible, imply the correctness of the construction.

Proving transparent hasher privacy is very simple: all information about $\boldsymbol{x}$ contained in d is masked by $u_0 + a \cdot u_1$. Since Power Ring-LWE implies Ring-LWE with respect to lattice distribution Γ, the latter is indistinguishable from random. Notice that all the information in hk is already contained in mk.

Also matrix privacy is easy to prove: under Power Ring-LWE, the elements $E_0, \ldots, E_{2n}$ are indistinguishable from random. So they hide all information about k. This ends the proof. □

By applying Theorem 7, we obtain the following corollary.

Corollary 3. *For any integer p, under Power Ring LWE with superpolynomial modulus-to-noise ratio, there exists half-chosen, semi-honest, non-interactive VOLE protocol over $\mathbb{Z}_p$ where the communication complexity is $O(\sqrt{n})$.*

7 Succinct HSS

In this section, we show how bilinear HSS schemes for multiples of the identity can be used to obtain particularly efficient 2-party HSS schemes for NC_1 circuits where the encoding of some of the inputs can be compressed into small digests. We call such primitive a *succinct HSS scheme*. Specifically, for any NC_1 circuit C mapping a vector in $\mathbb{Z}_q^m$ into a vector in $\mathbb{Z}_q^n$, a succinct HSS schemes allows two parties Alice and Bob, having inputs $\boldsymbol{x} \in \mathbb{Z}_q^m$ and $\boldsymbol{y} \in \mathbb{Z}_q^n$ respectively, to obtain a secret-sharing of the inner product $\langle C(\boldsymbol{x}), \boldsymbol{y} \rangle$ using a single round of interaction where the communication of Bob is independent of n. In other words, we obtain 2-party non-interactive protocols for particular classes of functions that achieve sublinear communication in the size of the inputs.

Special RMS Programs. In order to explain our idea, we need to first introduce the notion of *special restricted multiplication straightline (RMS) program*. We recall that the class of RMS programs over a ring R consists of all arithmetical circuits over R with fan-in 2, where linear operation are free, but multiplications require at least one of the factors to be an input wire. We also recall that all circuits in NC_1 can be represented as an RMS program of polynomial size [BGI16, Appendix A].

Special RMS programs generalise what we just explained. We will split the input wires into two classes: *standard inputs* and *special inputs*. Linear operations are again free, however, multiplications not only require at least one of the factors to be an input wire, but they also require the latter to be of standard type. The only operations allowed on special inputs are therefore linear gates or multiplications by a standard input. In other words, despite being input wires, special inputs are treated as any other internal wire of the RMS program. The latter are usually referred to as *memory wires*.

Definition 22 (Special RMS Program). *A special restricted multiplication straightline program (RMS) consists of a bound $B \in \mathbb{N}$, a modulus $q \in \mathbb{N}$ such that $B \leq q$ and an arithmetic circuit C with unbounded fan-out where the inputs wires are divided into two classes: standard inputs and special inputs. Furthermore, the only allowed gate types are the following:*

- $\mathsf{ConvertInput}(I_x) \to M_x$. *Load the value of the standard input wire I_x to the memory wire M_x.*
- $\mathsf{ConvertSpecialInput}(S_x) \to M_x$. *Load the value of the special input wire S_x to the memory wire M_x.*
- $\mathsf{Add}(M_x, M_y) \to M_z$. *Add the values of the memory wires M_x and M_y and store the result in the memory wire M_z.*
- $\mathsf{Mult}(I_x, M_y) \to M_z$. *Multiply the values of the standard input wire I_x by the value of the memory wire M_y. Store the result in the memory wire M_z.*
- $\mathsf{Output}(M_z) \to z$. *Output the value of the memory wire M_z reducing it modulo q.*

All inputs must belong to $\mathbb{Z}$. If the absolute value of any wire exceeds the bound B, the output of the evaluation is $\perp$.

In the following lemma, we prove that if C is in NC_1, then the function $\langle C(\boldsymbol{x}), \boldsymbol{y}\rangle$ can be represented as a special RMS program where $\boldsymbol{y}$ are special inputs.

Lemma 1. *Let R be a ring and let C be an RMS program over R taking inputs in R^m and outputting in R^n. Then, the function $f : R^m \times R^n \to R$ that maps $(\boldsymbol{x}, \boldsymbol{y})$ into $\langle C(\boldsymbol{x}), \boldsymbol{y}\rangle$ is computable by a special RMS program C' where $\boldsymbol{x}$ is the standard input and $\boldsymbol{y}$ is the special input.*

Proof. For every $i \in [n]$, let C_i be the subprogram of C that computes the i-th element of the output. We consider the program C'_i obtained by modifying C_i as follows:

- the program starts by loading the special input y_i into a memory wire,
- every addition gate, output gate and multiplication gate are left untouched,
- every input conversion gate is substituted by a multiplication of the input by y_i
- every addition of a constant $k \in R$ is substituted by an addition of $k \cdot y_i$.

We claim that C'_i computes $y_i \cdot C_i(\boldsymbol{x})$. Indeed, if a memory wire stored a value z during the evaluation of $C_i(\boldsymbol{x})$, in the evaluation of C'_i, the wire stores $y_i \cdot z$. We prove this by induction. If the memory wire is the output of an input conversion gate, our claim is easily verified. Now, consider a multiplication gate. By inductive hypothesis, if in C_i the gate computed the product $x \cdot z$ where x comes from the input wire and z from the memory wire, in C'_i, we will compute $x \cdot (y_i \cdot z)$. So our claim holds also for the output wire of the multiplication. Moving on to addition gates, by inductive hypothesis, if in C_i the gate computed the sum $z_1 + z_2$, in C'_i, we will compute $(y_i \cdot z_1) + (y_i \cdot z_2)$. Again, our claim is verified. Finally, we consider additions by constants. If in C_i any of these gates computed the sum $z + k$ where z was the value of a memory wire and k a constant, in C'_i, we will compute $(y_i \cdot z) + (y_i \cdot k)$. To conclude, we know that $C'_i(\boldsymbol{x}, \boldsymbol{y})$ will compute $y_i \cdot C_i(\boldsymbol{x})$.

We therefore build C' by evaluating $C'_i(\boldsymbol{x}, \boldsymbol{y})$ for every $i \in [n]$ and then adding the results. □

Defining succinct HSS. Below, we formalise the definition of succinct HSS: a 2-party HSS scheme for the evaluation of special RMS programs, where the parties can compress the encodings of their special inputs into small digests. The primitive relies on a setup that takes as input an upper-bound n on the number of special inputs that can be hashed into the same digest. In order to evaluate a special RMS program where the number of special inputs exceeds n, the parties simply need to send multiple digests.

Definition 23 (Succinct HSS). *A succinct HSS scheme over $\mathbb{Z}_q$ is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Input}, \mathsf{Eval})$ *with the following syntax:*

- Setup *is randomised and takes as input the security parameter $\mathbb{1}^\lambda$ and the input length $\mathbb{1}^n$. The output is a public key pk and evaluation keys ek_0 and ek_1.*
- Hash *is randomised and takes as input a public key pk, $b \in \{0,1\}$ and an input $\boldsymbol{x} \in \mathbb{Z}^n$. The output is a digest d and the hasher secret information ψ.*
- Input *is randomised and takes as input a public key pk and a value $y \in \mathbb{Z}$. The output is an encoding I of the input.*
- Eval *is deterministic and takes as input the evaluation key ek, the description of a special RMS program f, standard input encodings $I_1, \ldots, I_m$, digests $d_1, \ldots, d_{\ell_0}$ and hasher secret information $\psi_1, \ldots, \psi_{\ell_1}$ for some $m, \ell_0, \ell_1 \in \mathbb{N}$. The output is an element $s \in \mathbb{Z}_q$.*

We require also the following properties

1. ***(Correctness).*** *For every $n, m, \ell_0, \ell_1 \in \mathbb{N}$, special inputs $\boldsymbol{x}^{\mathbf{0}}_{\mathbf{1}}, \ldots, \boldsymbol{x}^{\mathbf{0}}_{\boldsymbol{\ell_0}}, \boldsymbol{x}^{\mathbf{1}}_{\mathbf{1}}, \ldots, \boldsymbol{x}^{\mathbf{1}}_{\boldsymbol{\ell_1}} \in \mathbb{Z}^n$, standard input $\boldsymbol{y} \in \mathbb{Z}^m$ and special RMS program $f : \mathbb{Z}^{n\cdot(\ell_0+\ell_1)} \times \mathbb{Z}^m \to \mathbb{Z}_q$, the following probability must be negligible.*

$$\Pr\left[s_0 + s_1 \neq z \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ \forall b \in \{0,1\}, i \in [\ell_b] : (d^b_i, \psi^b_i) \xleftarrow{\$} \mathsf{Hash}(\mathsf{pk}, b, \boldsymbol{x}^{\boldsymbol{b}}_{\boldsymbol{i}}) \\ \forall j \in [m] : I_j \xleftarrow{\$} \mathsf{Input}(\mathsf{pk}, y_j) \\ s_0 \leftarrow \mathsf{Eval}(\mathsf{ek}_0, f, I_1, \ldots, I_m, d^1_1, \ldots, d^1_{\ell_1}, \psi^0_1, \ldots, \psi^0_{\ell_0}) \\ s_1 \leftarrow \mathsf{Eval}(\mathsf{ek}_1, f, I_1, \ldots, I_m, d^0_1, \ldots, d^0_{\ell_0}, \psi^1_1, \ldots, \psi^1_{\ell_1}) \\ z \leftarrow f(\boldsymbol{x}^{\mathbf{0}}_{\mathbf{1}}, \ldots, \boldsymbol{x}^{\mathbf{0}}_{\boldsymbol{\ell_0}}, \boldsymbol{x}^{\mathbf{1}}_{\mathbf{1}}, \ldots, \boldsymbol{x}^{\mathbf{1}}_{\boldsymbol{\ell_1}}, \boldsymbol{y}) \end{array} \right]$$

2. ***(Hasher Privacy).*** *For every $n \in \mathbb{N}$, $b \in \{0,1\}$ and special inputs $\boldsymbol{x_0}, \boldsymbol{x_1} \in \mathbb{Z}^n$, the two values $i \in \{0,1\}$ are computationally indistinguishable in the following distribution*

$$\left\{ (\mathsf{pk}, \mathsf{ek}_b, d) \middle| \begin{array}{l} (\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ (d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{pk}, 1-b, \boldsymbol{x_i}) \end{array} \right\}$$

3. ***(Input Privacy).*** *For every $n \in \mathbb{N}$, $b \in \{0,1\}$ and standard inputs $y_0, y_1 \in \mathbb{Z}$, the two values $i \in \{0,1\}$ are computationally indistinguishable in the following distribution*

$$\left\{ (\mathsf{pk}, \mathsf{ek}_b, I) \middle| \begin{array}{l} (\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n) \\ I \xleftarrow{\$} \mathsf{Input}(\mathsf{pk}, y_i) \end{array} \right\}$$

4. ***(Efficiency).*** *The size of the hash d is $\mathsf{poly}(\lambda, \log|R|) \cdot o(n)$.*

We highlight that, in succinct HSS schemes, we can evaluate special RMS programs on standard inputs provided by external entities (in particular, none of the HSS participants needs to know the values hidden in the encodings of these inputs). Special inputs, on the other hand, need to be provided by the participant as the private information used for the generation of the digests is later on needed in the evaluation.

Below, we formalise an additional property of succinct HSS schemes: we say that the scheme has pseudorandom shares if, as long as both evaluation keys are kept private, the shares produced by the evaluation

look random conditioned on adding up to the correct value. We highlight that any succinct HSS scheme can be easily converted into a succinct HSS scheme with pseudorandom shares by simply augmenting the evaluation keys with a PRF key. We use the latter to rerandomise the shares produced by the evaluation without having to interact.

Definition 24 (Pseudorandom Shares). *We say that an succinct HSS scheme* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Input}, \mathsf{Eval})$ *over* $\mathbb{Z}_q$ *has pseudorandom shares if, for every* $n, m, \ell_0, \ell_1 \in \mathbb{N}$*, special inputs* $\boldsymbol{x}_1^0, \ldots, \boldsymbol{x}_{\ell_0}^0, \boldsymbol{x}_1^1, \ldots, \boldsymbol{x}_{\ell_1}^1 \in \mathbb{Z}^n$*, standard input* $\boldsymbol{y} \in \mathbb{Z}^m$ *and special RMS program* $f : \mathbb{Z}^{n\cdot(\ell_0+\ell_1)} \times \mathbb{Z}^m \to \mathbb{Z}_q$*, the following distribution is indistinguishable from uniform.*

$$\left\{ s_0 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \xleftarrow{\$} \mathsf{Setup}(1\!\!1^\lambda, 1\!\!1^n) \\ \forall b \in \{0,1\}, i \in [\ell_b] : (d_i^b, \psi_i^b) \xleftarrow{\$} \mathsf{Hash}(\mathsf{pk}, b, \boldsymbol{x_i^b}) \\ \forall j \in [m] : I_j \xleftarrow{\$} \mathsf{Input}(\mathsf{pk}, y_j) \\ s_0 \leftarrow \mathsf{Eval}(\mathsf{ek}_0, f, I_1, \ldots, I_m, d_1^1, \ldots, d_{\ell_1}^1, \psi_1^0, \ldots, \psi_{\ell_0}^0) \end{array} \right\}$$

In many applications of succinct HSS, we need to rely on a trusted dealer that generates and distributes the evaluation keys to the parties. This is for instance the case in our sublinear communication MPC protocols for layered circuits. Our goal is however to achieve security in the plain model, so, we need to substitute such trusted setup with another MPC protocol and the latter needs low communication complexity. In particular, we desire the communication to be $O(n^2) \cdot \mathsf{poly}(\lambda)$ or even $o(n) \cdot \mathsf{poly}(\lambda)$, where n is the parameter given as input to the setup of succinct HSS.

Definition 25 (Setup Tightness and Setup Sublinearity). *We say that a succinct HSS scheme* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Input}, \mathsf{Eval})$ *has a tight setup if there exists a semi-honest protocol that implements the functionality* $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$ *(see Fig. 12) with* $O(B) \cdot \mathsf{poly}(\lambda)$ *communication complexity, where* B *denotes the size of the triple* $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1)$*. We say that the scheme satisfies setup sublinearity if there exists a semi-honest protocol that implements the functionality* $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$ *(see Fig. 12) with* $o(n) \cdot \mathsf{poly}(\lambda)$ *communication complexity, where* n *is the parameter given as input to* Setup*.*

The succinct HSS Setup $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$

On input (Init, n) from all parties, the functionality performs the following operations:

1. $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \xleftarrow{\$} \mathsf{HSS.Setup}(1\!\!1^\lambda, 1\!\!1^n)$
2. Provide $\mathcal{P}_0$ with $(\mathsf{pk}, \mathsf{ek}_0)$ and $\mathcal{P}_1$ with $(\mathsf{pk}, \mathsf{ek}_1)$.

Fig. 12. The succinct HSS Setup $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$

7.1 Building Succinct HSS using Bilinear HSS

We are ready to explain how we can build succinct HSS schemes: we present a compiler that, using a strongly hasher private bilinear HSS scheme, converts any HSS scheme for RMS programs satisfying particular properties into a succinct HSS scheme. Most known HSS constructions for RMS programs [BGI16,BKS19,OSY21, ADOS22] satisfy the conditions needed for compilation.

Special HSS schemes. We formalise the properties that the compiler requires from the HSS scheme for RMS programs. We call any construction of this type a *special HSS scheme*. Essentially, an HSS scheme is special if, during the evaluation, every memory wire of the evaluated RMS program is associated with at most an additive secret-sharing of the underlying value x and an additive secret-sharing of $\boldsymbol{k} \cdot x$ for some private key $\boldsymbol{k}$ that can be reconstructed from the evaluation keys. Given any special RMS program C, encodings of the standard inputs and secret-sharings of x and $\boldsymbol{k} \otimes x$ for every special input x, the scheme should allow to evaluate C on the inputs without any interaction.

Definition 26 (Special HSS). *Let q be a positive integer. A 2-party HSS scheme* $(\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ *over $\mathbb{Z}_q$ is special if there exist PPT algorithms* $(\mathsf{Key}, \mathsf{SpecialEval})$ *with the following syntax:*

- Key *is deterministic and takes as input the evaluation keys* ek_0 *and* ek_1*. The output is a secret key* $\boldsymbol{k} \in \mathbb{Z}_q^t$*.*
- $\mathsf{SpecialEval}$ *is deterministic and takes as input an evaluation key* ek_b*, a special RMS program* $f : \mathbb{Z}^n \times \mathbb{Z}^m \to \mathbb{Z}^n$ *for some* $m, n \in \mathbb{N}$ *such that* $f \in \mathcal{C}$*, standard input encodings* $I_1, \ldots, I_m$ *and values* $\boldsymbol{x_b} \in \mathbb{Z}_q^n$ *and* $\hat{\boldsymbol{x}}_{\boldsymbol{b}} \in \mathbb{Z}_q^{t \cdot n}$*. The output is a values* $s_b \in \mathbb{Z}_q$*.*

We require also the following properties:

- *for every* $n, m \in \mathbb{N}$*, special RMS program* $f : \mathbb{Z}^n \times \mathbb{Z}^m \to \mathbb{Z}_q$*, standard input* $\boldsymbol{y} \in \mathbb{Z}^m$*, special input* $\boldsymbol{x} \in \mathbb{Z}^n$ *and values* $\boldsymbol{x_0} \in \mathbb{Z}_q^n$ *and* $\hat{\boldsymbol{x}}_{\boldsymbol{0}} \in \mathbb{Z}_q^{t \cdot n}$*,*

$$\Pr\left[s_0 + s_1 \neq f(\boldsymbol{x}, \boldsymbol{y}) \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \xleftarrow{\$} \mathsf{Setup}(1\!\!1^\lambda) \\ \boldsymbol{k} \leftarrow \mathsf{Key}(\mathsf{ek}_0, \mathsf{ek}_1) \\ \boldsymbol{x_1} \leftarrow \boldsymbol{x} - \boldsymbol{x_0} \bmod q \\ \hat{\boldsymbol{x}}_{\boldsymbol{1}} \leftarrow \boldsymbol{k} \otimes \boldsymbol{x} - \hat{\boldsymbol{x}}_{\boldsymbol{0}} \bmod q \\ \forall j \in [m] : I_j \xleftarrow{\$} \mathsf{Input}(\mathsf{pk}, y_j) \\ \forall b \in \{0,1\} : s_b \leftarrow \mathsf{SpecialEval}(\mathsf{ek}_b, f, I_1, \ldots, I_m, \boldsymbol{x_b}, \hat{\boldsymbol{x}}_{\boldsymbol{b}}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

- *for every* $n, m \in \mathbb{N}$*, special RMS program* $f : \mathbb{Z}^n \times \mathbb{Z}^m \to \mathbb{Z}_q$*, standard input* $\boldsymbol{y} \in \mathbb{Z}^m$ *and values* $\boldsymbol{x_0} \in \mathbb{Z}_q^n$ *and* $\hat{\boldsymbol{x}}_{\boldsymbol{0}} \in \mathbb{Z}_q^{t \cdot n}$*, the following distribution is computationally indistinguishable from uniform.*

$$\left\{ s_0 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \xleftarrow{\$} \mathsf{Setup}(1\!\!1^\lambda) \\ \forall j \in [m] : I_j \xleftarrow{\$} \mathsf{Input}(\mathsf{pk}, y_j) \\ s_0 \leftarrow \mathsf{SpecialEval}(\mathsf{ek}_0, f, I_1, \ldots, I_m, \boldsymbol{x_0}, \hat{\boldsymbol{x}}_{\boldsymbol{0}}) \end{array} \right\}$$

Almost all known HSS constructions for RMS programs satisfy the properties of Def. 26. The only exception is the DDH-based construction of [BGI16], which suffers from a non-negligible correctness error.

Theorem 10 ([OSY21, ADOS22, BKS19]). *The following hold:*

- *Under DCR over the Paillier group $\mathbb{Z}_{N^2}^*$, there exists a special 2-party HSS scheme over $\mathbb{Z}_N$.*
- *Under DDH, the hidden subgroup assumption and the small exponent assumption over class groups, there exists a special 2-party HSS scheme over $\mathbb{Z}_q$ for any prime number $q = O(2^\lambda)$.*
- *Under LWE with superpolynomial modulus-to-noise ratio, there exists a special 2-party HSS scheme over $\mathbb{Z}_p$ for any $p \in \mathbb{N}$.*

The succinct HSS compiler. We finally formalise our compiler. Notice that a special HSS scheme already almost allows evaluating special RMS programs: the only thing that we are missing is a mechanism to non-interactively provide secret-sharings of $\boldsymbol{k} \cdot x$ for every special input x, without leaking any information about the secret-key $\boldsymbol{k}$. We further desire this mechanism to require sublinear communication in the size of the special inputs.

Luckily, bilinear HSS schemes for multiples of the identity give exactly what we need: we generate random vectors $\boldsymbol{k_0}$ and $\boldsymbol{k_1}$ such that $\boldsymbol{k_0} + \boldsymbol{k_1} = \boldsymbol{k}$. Then, we encode each entry of $\boldsymbol{k_0}$ using the bilinear HSS scheme. We repeat the operation for $\boldsymbol{k_1}$. In this way, we obtain vectors of matrix encodings and corresponding secret information $(\boldsymbol{E_0}, \boldsymbol{\phi_0})$ and $(\boldsymbol{E_1}, \boldsymbol{\phi_1})$. We include $\boldsymbol{E_0}$ and $\boldsymbol{E_1}$ as part of the HSS public key. We include instead $\boldsymbol{k_0}$ and $\boldsymbol{\phi_0}$ in the first party's evaluation key and $\boldsymbol{k_1}$ and $\boldsymbol{\phi_1}$ in the second party's evaluation key. Notice that if the first party now sends a bilinear HSS digest of its special inputs $\boldsymbol{x}$, the players can immediately obtain a secret-sharing of $\boldsymbol{k_1} \otimes \boldsymbol{x}$. In order to convert the latter in a secret-sharing of $\boldsymbol{k} \cdot \boldsymbol{x}$, the first party just needs to add $\boldsymbol{k_0} \otimes \boldsymbol{x}$ to its share.

Theorem 11. *Assume the existence of a 2-party bilinear HSS scheme for the matrix class $\mathcal{M} \supseteq \mathcal{K}$ satisfying strong hasher privacy. Suppose also a 2-party special HSS scheme, both over $\mathbb{Z}_q$. Then, there exists a succinct HSS scheme over $\mathbb{Z}_q$ with pseudorandom shares.*

Moreover, if the bilinear HSS scheme satisfies transparent hasher privacy, the succinct HSS scheme has a tight setup. If the bilinear HSS scheme satisfies transparent hasher privacy, key-compactness and matrix-compactness, the succinct HSS scheme has a sublinear setup.

Proof. Let $\mathsf{BHSS} := (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Matrix}, \mathsf{HasherEval}, \mathsf{MatrixEval})$ be the bilinear HSS scheme and let $\mathsf{HSS} := (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval}, \mathsf{Key}, \mathsf{SpecialEval})$ be the special HSS scheme. The succinct HSS scheme is described in Fig. 13.

SUCCINCT HSS SCHEME

Let $L_1(\lambda, n)$ and $L_2(\lambda, n)$ be the length of the randomness needed by $\mathsf{BHSS.Setup}$ and $\mathsf{BHSS.Matrix}$.

$\mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$

1. $(\mathsf{pk}', \mathsf{ek}'_0, \mathsf{ek}'_1) \xleftarrow{\$} \mathsf{HSS.Setup}(\mathbb{1}^\lambda)$
2. $\boldsymbol{k} \leftarrow \mathsf{HSS.Key}(\mathsf{ek}'_0, \mathsf{ek}'_1)$
3. $\boldsymbol{k_0} \xleftarrow{\$} \mathbb{Z}_q^t$
4. $\boldsymbol{k_1} \leftarrow \boldsymbol{k} - \boldsymbol{k_0}$
5. $\forall b \in \{0,1\} : \boldsymbol{r_b} \xleftarrow{\$} \{0,1\}^{L_1(\lambda,n)}$
6. $\forall j \in [t], b \in \{0,1\} : \boldsymbol{r_{b,j}} \xleftarrow{\$} \{0,1\}^{L_2(\lambda,n)}$
7. $\forall b \in \{0,1\} : (\mathsf{hk}_b, \mathsf{mk}_b) \leftarrow \mathsf{BHSS.Setup}(\mathbb{1}^\lambda, \mathbb{1}^n; \boldsymbol{r_b})$
8. $\forall j \in [t], b \in \{0,1\} : (E_{b,j}, \phi_{b,j}) \leftarrow \mathsf{BHSS.Matrix}(\mathsf{mk}_b, k_{b,j}; \boldsymbol{r_{b,j}})$
9. $\mathsf{pk} \leftarrow (\mathsf{pk}', \mathsf{hk}_0, \mathsf{hk}_1, E_{0,1}, \ldots, E_{0,t}, E_{1,1}, \ldots, E_{1,t})$
10. $\forall b \in \{0,1\} : \mathsf{ek}_b \leftarrow (b, \boldsymbol{k_b}, \boldsymbol{r_b}, \boldsymbol{r_{b,1}}, \ldots, \boldsymbol{r_{b,t}}, \mathsf{ek}'_b, \phi_{b,1}, \ldots, \phi_{b,t}, \mathsf{mk}_b)$
11. Output pk, ek_0 and ek_1.

$\mathsf{Hash}(\mathsf{pk} = (\mathsf{pk}', \mathsf{hk}_0, \mathsf{hk}_1, (E_{0,j}, E_{1,j})_{j\in[t]}), b, \boldsymbol{x})$

1. $(d, \widehat{\psi}) \leftarrow \mathsf{BHSS.Hash}(\mathsf{hk}_{1-b}, \boldsymbol{x})$
2. Output d and $\psi := (\boldsymbol{x}, \widehat{\psi})$

$\mathsf{Input}(\mathsf{pk} = (\mathsf{pk}', \mathsf{hk}_0, \mathsf{hk}_1, (E_{0,j}, E_{1,j})_{j\in[t]}), y)$

1. Output $\mathsf{HSS.Input}(\mathsf{pk}', y)$

$\mathsf{Eval}\left(\mathsf{ek}, f, I_1, \ldots, I_m, d_1^{1-b}, \ldots, d_{\ell_{1-b}}^{1-b}, \psi_1^b, \ldots, \psi_{\ell_b}^b\right)$

1. rewrite ek as $(b, \boldsymbol{k_b}, \boldsymbol{r_b}, \boldsymbol{r_{b,1}}, \ldots, \boldsymbol{r_{b,t}}, \mathsf{ek}'_b, \phi_{b,1}, \ldots, \phi_{b,t}, \mathsf{mk}_b)$
2. $\forall i \in [\ell_b]$, rewrite ψ_i^b as $(\boldsymbol{x_i^b}, \widehat{\psi}_i^b)$
3. $\forall i \in [\ell_b] : \boldsymbol{x_{i,b}^b} \leftarrow \boldsymbol{x_i^b}$
4. $\forall i \in [\ell_{1-b}] : \boldsymbol{x_{i,b}^{1-b}} \leftarrow \boldsymbol{0}$
5. $\forall i \in [\ell_b], j \in [t] : \hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{b}}[j] \leftarrow \boldsymbol{k_{b,j}} \cdot \boldsymbol{x_i^b} + \mathsf{BHSS.HasherEval}(\mathsf{hk}_{1-b}, E_{1-b,j}, \widehat{\psi}_i^b)$
6. $\forall i \in [\ell_{1-b}], j \in [t] : \hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{1-b}}[\boldsymbol{j}] \leftarrow -\mathsf{BHSS.MatrixEval}(\mathsf{mk}_b, d_i^{1-b}, \phi_{b,j})$
7. $\forall i \in [\ell_b] : \hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{b}} \leftarrow (\hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{b}}[\boldsymbol{1}], \ldots, \hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{b}}[\boldsymbol{t}])$
8. $\forall i \in [\ell_{1-b}] : \hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{1-b}} \leftarrow (\hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{1-b}}[\boldsymbol{1}], \ldots, \hat{\boldsymbol{x}}_{\boldsymbol{i,b}}^{\boldsymbol{1-b}}[\boldsymbol{t}])$
9. $\hat{\boldsymbol{x}}_{\boldsymbol{b}} \leftarrow (\hat{\boldsymbol{x}}_{\boldsymbol{1,b}}^{\boldsymbol{0}}, \ldots, \hat{\boldsymbol{x}}_{\boldsymbol{\ell_0,b}}^{\boldsymbol{0}}, \hat{\boldsymbol{x}}_{\boldsymbol{1,b}}^{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{x}}_{\boldsymbol{\ell_1,b}}^{\boldsymbol{1}})$
10. $\boldsymbol{x_b} \leftarrow (\boldsymbol{x_{1,b}^0}, \ldots, \boldsymbol{x_{\ell_0,b}^0}, \boldsymbol{x_{1,b}^1}, \ldots, \boldsymbol{x_{\ell_1,b}^1})$
11. $s_b \leftarrow \mathsf{HSS.SpecialEval}(\mathsf{ek}'_b, f, I_1, \ldots, I_m, \boldsymbol{x_b}, \hat{\boldsymbol{x}}_{\boldsymbol{b}})$
12. Output s_b.

Fig. 13. succinct HSS Scheme

We prove our claim starting from correctness. By the correctness of BHSS, we have that, for every $i \in [\ell_0]$ and $j \in [t]$,

$$\begin{aligned}\hat{\boldsymbol{x}}^{\boldsymbol{0}}_{\boldsymbol{i},\boldsymbol{0}}[\boldsymbol{j}] + \hat{\boldsymbol{x}}^{\boldsymbol{0}}_{\boldsymbol{i},\boldsymbol{1}}[\boldsymbol{j}] &= k_{j,0} \cdot \boldsymbol{x}^{\boldsymbol{0}}_{\boldsymbol{i}} + \mathsf{BHSS.HasherEval}(\mathsf{hk}_1, E_{1,j}, \widehat{\psi}^0_i) \\ &\quad - \mathsf{BHSS.MatrixEval}(\mathsf{mk}_1, d^0_i, \phi_{1,j}) \\ &= k_{j,0} \cdot \boldsymbol{x}^{\boldsymbol{0}}_{\boldsymbol{i}} + k_{j,1} \cdot \boldsymbol{x}^{\boldsymbol{0}}_{\boldsymbol{i}} = k_j \cdot \boldsymbol{x}^{\boldsymbol{0}}_{\boldsymbol{i}}.\end{aligned}$$

In an analogous way, we have that for every $i \in [\ell_1]$ and $j \in [t]$,

$$\begin{aligned}\hat{\boldsymbol{x}}^{\boldsymbol{1}}_{\boldsymbol{i},\boldsymbol{0}}[\boldsymbol{j}] + \hat{\boldsymbol{x}}^{\boldsymbol{1}}_{\boldsymbol{i},\boldsymbol{1}}[\boldsymbol{j}] &= k_{j,1} \cdot \boldsymbol{x}^{\boldsymbol{1}}_{\boldsymbol{i}} + \mathsf{BHSS.HasherEval}(\mathsf{hk}_0, E_{0,j}, \widehat{\psi}^1_i) \\ &\quad - \mathsf{BHSS.MatrixEval}(\mathsf{mk}_0, d^1_i, \phi_{0,j}) \\ &= k_{j,1} \cdot \boldsymbol{x}^{\boldsymbol{1}}_{\boldsymbol{i}} + k_{j,0} \cdot \boldsymbol{x}^{\boldsymbol{1}}_{\boldsymbol{i}} = k_j \cdot \boldsymbol{x}^{\boldsymbol{1}}_{\boldsymbol{i}}.\end{aligned}$$

In other words, $\hat{\boldsymbol{x}}_{\boldsymbol{0}} + \hat{\boldsymbol{x}}_{\boldsymbol{1}} = \boldsymbol{k} \otimes (\boldsymbol{x}_{\boldsymbol{0}} + \boldsymbol{x}_{\boldsymbol{1}}) = \boldsymbol{k} \otimes (\boldsymbol{x}^{\boldsymbol{0}}_{\boldsymbol{1}}, \ldots, \boldsymbol{x}^{\boldsymbol{0}}_{\boldsymbol{\ell_0}}, \boldsymbol{x}^{\boldsymbol{1}}_{\boldsymbol{1}}, \ldots, \boldsymbol{x}^{\boldsymbol{1}}_{\boldsymbol{\ell_1}})$. We conclude by relying on the correctness of the special HSS scheme HSS.

The pseudorandomness of the shares is immediately implied by the second property of special HSS schemes.

Hasher privacy is immediately implied by the strong hasher privacy of BHSS. Finally, we prove input privacy. Consider any $b \in \{0,1\}$. We rely on a sequence of indistinguishable hybrids to prove that no PPT adversary can distinguish $\mathsf{Input}(\mathsf{pk}, y_0)$ from $\mathsf{Input}(\mathsf{pk}, y_1)$ even when provided with pk and ek_b.

Hybrid 0. In this hybrid, we generate pk, ek_0 and ek_1 using $\mathsf{Setup}(\mathbb{1}^\lambda, \mathbb{1}^n)$.

Hybrid 1. In this hybrid, we change the distribution of $E_{1-b,j}$ for every $j \in [t]$: instead of computing $\mathsf{BHSS.Matrix}(\mathsf{mk}_{1-b}, k_{1-b,j})$, we generate $E_{1-b,j}$ by running $\mathsf{BHSS.Matrix}(\mathsf{mk}_{1-b}, 0)$. This hybrid is indistinguishable from the previous one due to the matrix privacy of BHSS. Observe that pk and ek_b are now independent of $\boldsymbol{k_{1-b}}$.

Hybrid 2. In this hybrid, instead of providing the adversary with $\mathsf{Input}(\mathsf{pk}, y_0)$ or $\mathsf{Input}(\mathsf{pk}, y_1)$, we provide it with the output of $\mathsf{Input}(\mathsf{pk}, 0)$.

Finally, we show that if the bilinear HSS scheme is key-compact and matrix-compact, the succinct HSS scheme satisfies setup sublinearity. We start by observing that there exists a semi-honest protocol $\Pi_{\mathsf{SpecialHSS\text{-}Setup}}$ that implements the functionality $\mathcal{F}_{\mathsf{SpecialHSS\text{-}Setup}}$ in Fig. 14 with $\mathsf{poly}(\lambda)$ communication. Our succinct HSS setup protocol will start by executing $\Pi_{\mathsf{SpecialHSS\text{-}Setup}}$. In this way, each party $\mathcal{P}_b$ obtains $(\mathsf{pk}', \mathsf{ek}'_b, \boldsymbol{k_b})$. Subsequently, $\mathcal{P}_b$ samples $\boldsymbol{r_b} \xleftarrow{\$} \{0,1\}^{L_1(\lambda,n)}$ and $\boldsymbol{r_{b,j}} \xleftarrow{\$} \{0,1\}^{L_2(\lambda,n)}$ for every $j \in [t]$. It then computes

$$(\mathsf{hk}_b, \mathsf{mk}_b) \leftarrow \mathsf{BHSS.Setup}(\mathbb{1}^\lambda, \mathbb{1}^n; \boldsymbol{r_b})$$

and, for every $j \in [t]$,

$$(E_{b,j}, \phi_{b,j}) \leftarrow \mathsf{BHSS.Matrix}(\mathsf{mk}_b, k_{b,j}; \boldsymbol{r_{b,j}}).$$

Finally, it sends $(\mathsf{hk}_b, E_{b,1}, \ldots, E_{b,j})$ to the other party. After receiving, $(\mathsf{hk}_{1-b}, E_{1-b,1}, \ldots, E_{1-b,j})$ from the other participant, party $\mathcal{P}_b$ outputs $\mathsf{pk} := (\mathsf{pk}', \mathsf{hk}_0, \mathsf{hk}_1, E_{0,1}, \ldots, E_{0,t}, E_{1,1}, \ldots, E_{1,t})$ and $\mathsf{ek}_b := (b, \boldsymbol{k_b}, \boldsymbol{r_b}, \boldsymbol{r_{b,1}}, \ldots, \boldsymbol{r_{b,t}}, \mathsf{ek}'_b, \phi_{b,1}, \ldots, \phi_{b,t}, \mathsf{mk}_b)$. We observe that, thanks to transparent hasher privacy, this protocol implements $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$ (see Fig. 12) and the communication complexity is $O(B) \cdot \mathsf{poly}(\lambda)$. Moreover, if the bilinear HSS scheme is key-compact and matrix-compact, the communication complexity is $\mathsf{poly}(\lambda) + o(n) \cdot \mathsf{poly}(\lambda, \log q) + t \cdot o(n) \cdot \mathsf{poly}(\lambda, \log q)$. Since t and $\log q$ are polynomial in λ, our claim follows. This ends the proof. □

The following corollary summarises all known succinct HSS constructions.

Corollary 4. *The following hold:*

- *Under DCR over the Paillier group $\mathbb{Z}^*_{N^2}$, there exists a succinct HSS scheme over $\mathbb{Z}_N$ with pseudorandom shares and setup sublinearity where the public key size, evaluation key size and the hash size is $O(n^{2/3}) \cdot \mathsf{poly}(\lambda)$. If we additionally assume Power-DDH over $\mathbb{Z}^*_{N^2}$, there exists a succinct HSS scheme with pseudorandom shares and setup sublinearity where the public key size, evaluation key size and the hash size is $O(\sqrt{n}) \cdot \mathsf{poly}(\lambda)$.*

THE FUNCTIONALITY $\mathcal{F}_{\mathsf{SpecialHSS\text{-}Setup}}$

The functionality computes the following operations:

1. $(\mathsf{pk}', \mathsf{ek}'_0, \mathsf{ek}'_1) \xleftarrow{\$} \mathsf{HSS.Setup}(\mathbb{1}^\lambda)$
2. $\boldsymbol{k} \leftarrow \mathsf{HSS.Key}(\mathsf{ek}'_0, \mathsf{ek}'_1)$
3. $\boldsymbol{k_0} \xleftarrow{\$} \mathbb{Z}_q^t$
4. $\boldsymbol{k_1} \leftarrow \boldsymbol{k} - \boldsymbol{k_0}$
5. Provide $\mathcal{P}_0$ with $(\mathsf{pk}', \mathsf{ek}'_0, \boldsymbol{k_0})$, provide $\mathcal{P}_1$ with $(\mathsf{pk}', \mathsf{ek}'_1, \boldsymbol{k_1})$.

Fig. 14. The functionality $\mathcal{F}_{\mathsf{SpecialHSS\text{-}Setup}}$

- *For any prime $q = \Omega(2^\lambda)$, under DDH, the hidden subgroup assumption, the small exponent assumption, the $n^{1/3}$-ary EDDH assumption and the uniformity assumption over class groups, there exists a succinct HSS scheme over $\mathbb{Z}_q$ with pseudorandom shares and setup sublinearity where the public key size, evaluation key size and the hash size is $O(n^{2/3}) \cdot \mathsf{poly}(\lambda)$. If we additionally assume Power-DDH over class groups, there exists a succinct HSS scheme with pseudorandom shares and setup sublinearity where the public key size, evaluation key size and the hash size is $O(\sqrt{n}) \cdot \mathsf{poly}(\lambda)$.*
- *For any integer p, under LWE with superpolynomial modulus-to-noise ratio, there exists a succinct HSS scheme over $\mathbb{Z}_p$ with pseudorandom shares and setup sublinearity where the evaluation key size and the hash size is $O(n^{2/3}) \cdot \mathsf{poly}(\lambda)$. Furthermore, under Power Ring-LWE with superpolynomial modulus-to-noise ratio, there exists a succinct HSS scheme over $\mathbb{Z}_p$ with pseudorandom shares and setup sublinearity where the evaluation key size and the hash size is $O(\sqrt{n}) \cdot \mathsf{poly}(\lambda)$.*

On Succinct HSS over $\mathbb{Z}_2$. Any 2-party succinct HSS scheme with pseudorandom shares over $\mathbb{Z}_q$, where q is superpolynomial in the security parameter can be easily converted into a succinct HSS scheme with pseudorandom shares over $\mathbb{Z}_2$. The conversion comes at no additional cost. In order to evaluate a binary circuit, we just evaluate it modulo q. At the end, both parties reduce their shares modulo 2 and, if q is odd, the first party also flips its value.

The reason why this trick works is that q is sufficiently large to not create any wrap-arounds during the evaluation of the circuit. Indeed, since RMS programs required that at least one of the inputs of every multiplication gate is actually an input to the whole circuit (and the latter is necessarily a bit), multiplications do not increase the magnitude of the values in the internal wires. Also, since the RMS program is over $\mathbb{Z}_2$, all linear operations correspond to additions of subsets of wires (multiplying by a constant over $\mathbb{Z}_2$ is either a multiplication by 1 or by 0). In conclusion, during the evaluation, the maximum value that a wire can assume is the size of the program, which is polynomial in λ.

We conclude that, at the end of the evaluation, the parties obtain an additive secret-sharing modulo q of a value $0 \leq y < q$ such that $|y| \leq \mathsf{poly}(\lambda)$. The output of the evaluation of the program over $\mathbb{Z}_2$ corresponds to $y \bmod 2$. Let $0 \leq y_0, y_1 < q$ be the two shares. Since the succinct HSS scheme has pseudorandom shares and q is superpolynomial, the probability that $y_0 < y$ is negligible. Therefore, with overwhelming probability, we have that $y_0 + y_1 = q + y$, where the addition is here performed over the integers. By reducing both sides modulo 2, we obtain that

$$(y_0 \bmod 2) \oplus (y_1 \bmod 2) = (y \bmod 2) \oplus (q \bmod 2).$$

This is exactly what we wanted.

8 Multiparty Distributed Point Function

We now show how 2-party succinct HSS can be used to build an N-party DPF for any $N = \mathsf{poly}(\lambda)$. We do this by presenting a semi-honest protocol that allows N parties holding additive secret-sharings of the bits of

$\alpha \in [n]$ and a multiplicative secret sharing of $\beta \in \mathbb{Z}_q^*$, to obtain an additive secret sharing of the unit vector

$$(\underbrace{0, \ldots, 0, \beta}_{\alpha}, 0, \ldots, 0) \in \mathbb{Z}_q^n$$

with $o(n)$ communication. It is easy to observe that such protocol leads immediately to an N-party DPF construction: in order to generate the DPF keys for a unit vector with special position $\alpha \in [2^n]$ and non-zero value $\beta \in \mathbb{Z}_q^*$, we run the protocol on input a random additive secret-sharing of the bits of α and a random multiplicative secret-sharing of β. The DPF key of any party $\mathcal{P}_i$ will consist of $\mathcal{P}_i$'s view in the protocol execution.

The idea behind the protocol is rather simple: we rely on a trusted setup that provides each pair of parties with the evaluation keys of a different 2-party succinct HSS instance. The execution will be started by party $\mathcal{P}_1$, which will define the vector $\boldsymbol{s_{1,2}}$ as the unit vector $\boldsymbol{u_{\alpha_1,\beta_1}}$ having special position α_1 (i.e. $\mathcal{P}_1$'s share of α) and non-zero value β_1 (i.e. $\mathcal{P}_1$'s multiplicative share of β).

We then enter a loop whose iterations are indexed by $j = 2, \ldots, N$. Each party $\mathcal{P}_i$ will participate in the protocol only in the iterations indexed by $j \geq i$. In particular, party $\mathcal{P}_1$ will be involved in all iterations, while party $\mathcal{P}_N$ only in the last one. In the iteration indexed by j, each party $\mathcal{P}_i$ with $i < j$ will obtain a vector $\boldsymbol{s_{i,j+1}} \in \mathbb{Z}_q^{2^n}$. We will satisfy the invariant

$$\sum_{i=1}^{j} \boldsymbol{s_{i,j+1}} = \boldsymbol{u_{\alpha'_j,\beta'_j}} \tag{1}$$

where $\alpha'_j := \bigoplus_{i=1}^{j} \alpha_i$ and $\beta'_j := \prod_{i=1}^{j} \beta_i$.

In the iteration indexed by j, party $\mathcal{P}_j$ will input the bits of α_j (i.e. $\mathcal{P}_j$'s share of α) and β_j (i.e. $\mathcal{P}_j$'s multiplicative share of β) in all succinct HSS execution where, on the other end, there is a party $\mathcal{P}_i$ with $i < j$. On the other side, $\mathcal{P}_i$ will send a digest of the vector $\boldsymbol{s_{i,j}}$. Then, $\mathcal{P}_i$ and $\mathcal{P}_j$ will evaluate a special RMS program that multiplies all the entries of $\boldsymbol{s_{i,j}}$ by β_j and shuffles them according to α_j (i.e., entry x will be moved to entry $x \oplus \alpha_j$). The share of the output obtained by party $\mathcal{P}_i$ will be the vector $\boldsymbol{s_{i,j+1}}$. Party $\mathcal{P}_j$ will instead obtain $\boldsymbol{s_{j,j+1}}$ by adding the shares it obtained in the j succinct HSS executions. It is easy to see that the invariant (1) is preserved. So, at the end of the protocol, the vectors $\boldsymbol{s_{i,N+1}}$ will consist of secret-sharing of the desired unit-vector.

Theorem 12. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Input}, \mathsf{Eval})$ *be a succinct HSS scheme over* $\mathbb{Z}_q$ *and let* F *be a PRF. Then, the protocol in Fig. 15 UC-implements the functionality* $\mathcal{F}_{\mathsf{DPF}}$ *(see Fig. 16) in the* $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$*-hybrid model (see Fig. 12) against static, semi-honest adversaries corrupting up to* $N - 1$ *parties.*

Furthermore, consider the algorithm pair $\mathsf{DPF} = (\mathsf{Gen}, \mathsf{Eval})$ *where*

- Gen *takes as input* $\boldsymbol{\alpha} \in \{0,1\}^k$ *where* $k < \log n$ *and* $\beta \in \mathbb{Z}_q$ *and outputs keys* $\kappa_1, \ldots, \kappa_N$ *such that* κ_i *consists of the view*[13] *of party* $\mathcal{P}_i$ *in the protocol* Π_{DPF} *on input random* $(\boldsymbol{\alpha_j}, \beta_j)_{j \in [N]}$ *such that* $\bigoplus_{j=1}^{N} \boldsymbol{\alpha_j} = \boldsymbol{\alpha}$ *and* $\prod_{j=1}^{N} \beta_j = \beta$.
- Eval *takes as input a key* κ_i *and an index* $\ell \in [2^k]$*. It reruns* Π_{DPF} *from* $\mathcal{P}_i$*'s perspective using the transcript in* κ_i *and outputs the* ℓ*-th entry of* $\boldsymbol{s_{i,N+1}}$.

Then, DPF *is an* N*-party DPF over* $\mathbb{Z}_q^*$ [14] *with domain* $[2^k]$ *having key size* $N \cdot k \cdot \mathsf{poly}(\lambda, \log q) + O(N) \cdot (t_0 + t_1 + t_2)$*, where* $t_0(\lambda, n)$*,* $t_1(\lambda, n)$ *and* $t_2(\lambda, n)$ *are the public key size, the evaluation key size and the digest size of* HSS*, respectively.*

Proof. We start by proving correctness. We claim that, for every $j \in [N]$, we have

$$\sum_{i=1}^{j} \boldsymbol{s_{i,j+1}} = \boldsymbol{u_{\alpha'_j,\beta'_j}}$$

[13] The view of party $\mathcal{P}_i$ consists of K_i, $\boldsymbol{\alpha_i}$, β_i, $(\mathsf{pk}_{i,j}, \mathsf{ek}_{i,j})_{j>i}$ and $(\mathsf{pk}_{i,j}, \mathsf{ek}_{j,i})_{j<i}$ and the messages received by $\mathcal{P}_i$ in Π_{DPF}.

[14] The construction guarantees the secrecy of β as long as it belongs to $\mathbb{Z}_q^*$. The privacy of $\boldsymbol{\alpha}$ is always guaranteed.

THE N-PARTY DPF PROTOCOL Π_{DPF}

For every $\{i,j\} \subseteq [N]$, let $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}^{i,j}$ be a copy of $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$ to which only $\mathcal{P}_i$ and $\mathcal{P}_j$ have access.

Initialisation: On input (Init, n), each party $\mathcal{P}_i$ performs the following operations:

1. for every $j \neq i$, send (Init, n) to $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}^{i,j}$. Receive $(\mathsf{pk}_{i,j}, \mathsf{ek}_{i,j})$ (if $i < j$) or $(\mathsf{pk}_{i,j}, \mathsf{ek}_{j,i})$ (if $i > j$) as an answer ($\mathcal{P}_j$ will obtain the other half of the key).
2. $\boldsymbol{K_i} \xleftarrow{\$} \{0,1\}^\lambda$.

Input: On input $(\mathsf{Input}, \mathsf{id}, x)$, party $\mathcal{P}_i$ performs the following operations:

1. $\boldsymbol{r_j} \leftarrow F(\boldsymbol{K_i}, (\mathsf{id}, j))$
2. $\forall j < i : I_j^{\mathsf{id}} \leftarrow \mathsf{HSS.Input}(\mathsf{pk}_{i,j}, x; \boldsymbol{r_j})$
3. Send (id, I_j) to party $\mathcal{P}_j$ for every $j < i$.
4. Store $x, (I_j)_{j<i}$ under the identity id.

DPF: On input $(\mathsf{DPF}, (\mathsf{id}_{j,0}, \mathsf{id}_{j,1}, \ldots, \mathsf{id}_{j,k})_{j\in[N]})$ from the environment, for some $k \leq \log n$, the parties performs the following operations:

1. For every $i \in [N]$, $\mathcal{P}_i$ retrieves the values β_i and $(I_{j,i}^0)_{j<i}$ it stored under the identity $\mathsf{id}_{i,0}$. For every $b \in [k]$, $\mathcal{P}_i$ also retrieves the values $\alpha_{i,b}$ and $(I_{j,i}^b)_{j<i}$ it stored under the identity $\mathsf{id}_{i,b}$. Finally, it sets $\boldsymbol{\alpha_i} := (\alpha_{i,1}, \ldots, \alpha_{i,k})$.
2. For every $j < i$, and $b \in [0..k]$, $\mathcal{P}_i$ retrieves the value $I_{i,j}^b$ it received from $\mathcal{P}_j$ under the identity $\mathsf{id}_{j,b}$.
3. $\mathcal{P}_1$ sets $\boldsymbol{s_{1,2}} \leftarrow \boldsymbol{u_{\alpha_1,\beta_1}}$
4. for $j = 2, \ldots, N$:
   - each $\mathcal{P}_i$ such that $i < j$ computes:
     (a) $\boldsymbol{r_{i,j}} \leftarrow F\big(\boldsymbol{K_i}, (j, (\mathsf{id}_{\ell,0}, \mathsf{id}_{\ell,1}, \ldots, \mathsf{id}_{\ell,k})_{\ell\in[N]})\big)$
     (b) $(d_{i,j}, \psi_{i,j}) \leftarrow \mathsf{HSS.Hash}(\mathsf{pk}_{i,j}, 0, \boldsymbol{s_{i,j}}; \boldsymbol{r_{i,j}})$
     (c) Send $d_{i,j}$ to $\mathcal{P}_j$
     (d) $\boldsymbol{s_{i,j+1}} \leftarrow \mathsf{HSS.Eval}(\mathsf{ek}_{i,j}, f_k, I_{i,j}^1, \ldots, I_{i,j}^k, I_{i,j}^0, \psi_{i,j})$ (see below)
   - $\mathcal{P}_j$ computes:
     (a) Receive $d_{i,j}$ from $\mathcal{P}_i$ for every $i < j$
     (b) $\forall i < j : \boldsymbol{s_{j,j+1}} \leftarrow \sum_{i=1}^{j-1} \mathsf{HSS.Eval}(\mathsf{ek}_{j,i}, f_k, I_{i,j}^1, \ldots, I_{i,j}^k, I_{i,j}^0, d_{i,j})$ (see below)
5. Each party $\mathcal{P}_i$ outputs $\boldsymbol{s_{i,N+1}}$

The Special RMS Program f_k

Special Input: an 2^k-dimensional vector $\boldsymbol{v}$
Standard Input: bits $\alpha_1, \ldots, \alpha_k$ and an element $\beta \in \mathbb{Z}$

1. $\boldsymbol{v_0} \leftarrow \boldsymbol{v}$
2. For $i = 1, \ldots, k$:
   (a) $\forall j \in [0..2^k - 1] : \boldsymbol{v_i}[j] \leftarrow \boldsymbol{v_{i-1}}[j] + \alpha_i \cdot (\boldsymbol{v_{i-1}}[j \oplus 2^{k-i}] - \boldsymbol{v_{i-1}}[j])$
3. $\forall j \in [0..2^k - 1] : \boldsymbol{w}[j] \leftarrow \boldsymbol{v_k}[j] \cdot \beta$
4. Output $\boldsymbol{w}$

Fig. 15. The N-Party DPF Protocol

THE N-PARTY DPF FUNCTIONALITY $\mathcal{F}_{\mathsf{DPF}}$

Initialisation: On input (Init, n) from each party, the functionality activates and stores n. Subsequent calls to this procedure are ignored. Let ι denote the index of a fixed honest party.

Input: On input $(\mathsf{Input}, \mathsf{id}, x)$ from a party $\mathcal{P}_i$ where $x \in \mathbb{Z}_q$, store x under the identity id.

DPF: On input $(\mathsf{DPF}, (\mathsf{id}_{j,0}, \mathsf{id}_{j,1}, \ldots, \mathsf{id}_{j,k})_{j\in[N]})$ from each party $\mathcal{P}_i$ for some $k \leq \log n$, the functionality performs the following operations:

1. For every $i \in [N]$, retrieve the value β_i stored under the identity $\mathsf{id}_{i,0}$.
2. For every $i \in [N]$ and $b \in [k]$, retrieve the value $\alpha_{i,b}$ stored under the identity $\mathsf{id}_{i,b}$. Let $\boldsymbol{\alpha_i} := (\alpha_{i,1}, \ldots, \alpha_{i,k})$.
3. $\boldsymbol{\alpha} \leftarrow \bigoplus_{i=1}^{N} \boldsymbol{\alpha_i}$
4. $\beta \leftarrow \prod_{i=1}^{N} \beta_i$
5. Receive $\boldsymbol{s_i} \in \mathbb{Z}_q^{2^k}$ from the adversary for every corrupted party $\mathcal{P}_i$
6. For every honest party $\mathcal{P}_i$ with $i \neq \iota$, $\boldsymbol{s_i} \xleftarrow{\$} \mathbb{Z}_q^{2^k}$
7. $\boldsymbol{s_\iota} \leftarrow \boldsymbol{u_{\alpha,\beta}} - \sum_{i\neq\iota} \boldsymbol{s_i}$.
8. Output $\boldsymbol{s_i}$ to every party $\mathcal{P}_i$.

Fig. 16. The N-Party DPF Functionality $\mathcal{F}_{\mathsf{DPF}}$

where $\boldsymbol{\alpha'_j} := \bigoplus_{i=1}^{j} \boldsymbol{\alpha_i}$ and $\beta'_j = \prod_{i=1}^{j} \beta_i$. The claim is clearly true for $j = 1$. We prove by induction that if the claim is true for $j-1$, then the claim is true also for j. We observe that, by the correctness of HSS, we have

$$\begin{aligned}
\sum_{i=1}^{j} \boldsymbol{s_{i,j+1}} &= \sum_{i=1}^{j-1} \boldsymbol{s_{i,j+1}} + \boldsymbol{s_{j,j+1}} \\
&= \sum_{i=1}^{j-1} \mathsf{HSS.Eval}(\mathsf{ek}_{i,j}, f_k, I^1_{i,j}, \ldots, I^k_{i,j}, I^0_{i,j}, \psi_{i,j}) \\
&+ \sum_{i=1}^{j-1} \mathsf{HSS.Eval}(\mathsf{ek}_{j,i}, f_k, I^1_{i,j}, \ldots, I^k_{i,j}, I^0_{i,j}, d_{i,j}) \\
&= \sum_{i=1}^{j-1} f_k(\boldsymbol{s_{i,j}}, \alpha^1_j, \ldots, \alpha^k_j, \beta_j)
\end{aligned}$$

We also observe that f_k is linear in the first entry (it only permutes the entries of the vector and multiplies all of them by the same element). So,

$$\begin{aligned}
\sum_{i=1}^{j} \boldsymbol{s_{i,j+1}} &= \sum_{i=1}^{j-1} f_k(\boldsymbol{s_{i,j}}, \alpha^1_j, \ldots, \alpha^k_j, \beta_j) \\
&= f_k\left(\sum_{i=1}^{j-1} \boldsymbol{s_{i,j}}, \alpha^1_j, \ldots, \alpha^k_j, \beta_j\right) \\
&= f_k\left(\boldsymbol{u}_{\alpha'_{j-1},\beta'_{j-1}}, \alpha^1_j, \ldots, \alpha^k_j, \beta_j\right) = \boldsymbol{u_{\alpha'_j,\beta'_j}}.
\end{aligned}$$

Next, we prove security. We consider the simulator $\mathcal{S}$ that just runs the protocol using dummy inputs for the honest parties, forwarding the inputs of the corrupted players to $\mathcal{F}_{\mathsf{DPF}}$ and sends $\boldsymbol{s_{i,N+1}}$ to $\mathcal{F}_{\mathsf{DPF}}$ for every corrupted party $\mathcal{P}_i$. We show that no semi-honest adversary can distinguish between real world and ideal world using a sequence of indistinguishable hybrids.

Hybrid 0. This hybrid corresponds to the real world.

Hybrid 1. In this hybrid, we fix an honest party $\mathcal{P}_\iota$. We compute the output of $\mathcal{P}_\iota$ as $\boldsymbol{u_{\alpha,\beta}} - \sum_{i\neq\iota} \boldsymbol{s_{i,N+1}}$. By the correctness of the protocol, this hybrid is indistinguishable from the previous one.

Hybrid 2. In this hybrid, for every honest party $\mathcal{P}_i$, we do not evaluate F on input K_i anymore. We instead sample the outputs at random. This hybrid is indistinguishable from the previous one due to the security of the PRF.

Hybrid 3. In this hybrid, for every honest party $\mathcal{P}_i$, instead of inputting $\alpha_i^1, \ldots, \alpha_i^k$ and β_i in the succinct HSS scheme with any other party $\mathcal{P}_j$, we input 0. This hybrid is indistinguishable from the previous one due to the input privacy of succinct HSS.

Hybrid 4. In this hybrid, for every honest party $\mathcal{P}_i$, instead of hashing $\boldsymbol{s_{i,j}}$ for every j, we hash $\mathbf{0}$. This hybrid is indistinguishable from the previous one thanks to the hasher privacy of the succinct HSS scheme.

Hybrid 5. This hybrid corresponds to the ideal world. What changed compared to Hybrid 3 is that the shares of the honest parties are now random conditioned on adding up to the right unit vector. This hybrid is indistinguishable from the previous one thanks to the fact that our succinct HSS scheme has pseudorandom shares.

The fact that $\mathsf{DPF} = (\mathsf{Gen}, \mathsf{Eval})$ is a DPF is an immediate consequence of the security of the protocol. This ends the proof. □

8.1 Sublinear Communication N-party Computation from N-party Distributed Point Functions

In this subsection, we show how our N-party DPF can be used to build N-party computation protocols for layered circuits achieving sublinear communication in the circuit size. We recall that layered circuits are boolean circuits where the nodes of the underlying graph can be partition into subsets $L_1, \ldots, L_d$ called layers and each wire of the circuit goes from a gate in some layer L_i to a gate in the layer L_{i+1}. The width of the layer coincides with its cardinality. The number of layers is called the depth of the circuit.

We achieve our result following the blueprint of Couteau [Cou19], who showed how to build sublinear communication MPC in the correlated randomness model. We recall his main result below.

Theorem 13 ([Cou19]). *Assume the existence of authenticated point-to-point channels. For any N-party functionality $\mathcal{F}$ represented by a layered boolean circuit C of size s with k inputs and m outputs, there exists a protocol that perfectly realises $\mathcal{F}$ against semi-honest adversaries corrupting up to $N-1$ parties in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model (see Fig. 17) with $O(k + N \cdot (m + s/\log\log s))$ communication, polynomial storage and needing a single call to $\mathcal{F}_{\mathsf{Prep}}$.*

To obtain our sublinear communication MPC protocol, we just need to implement the functionality $\mathcal{F}_{\mathsf{Prep}}$ with sublinear communication in the size of the evaluated circuit. We observe that $\mathcal{F}_{\mathsf{Prep}}$ is essentially generating B one-time-truth-tables [IKM+13], one for each block of the circuit. We recall that a one-time-truth-table for a function $f : \{0,1\}^n \to \{0,1\}^m$ consists of a secret sharing of a random shift $\boldsymbol{r} \in \{0,1\}^n$ and a secret-sharing of the truth table of $f_j^{\boldsymbol{r}}(\boldsymbol{x}) := f_j(\boldsymbol{x} \oplus \boldsymbol{r}[S_j])$ for every $j \in [m]$. Here, S_j consists of the subset of input bits upon which the j-th output bit of f may depend. The function $f_j : \{0,1\}^{|S_j|} \to \{0,1\}$ computes the j-th output bit of f.

We observe that N-party DPFs can be used to easily generate N-party one-time-truth-tables for any $O(\log\lambda)$-local function $f : \{0,1\}^n \to \{0,1\}^m$ (i.e. any function for which each output bit depends on at most $O(\log\lambda)$ input bits). The idea is the following: the parties pick the secret-sharing of a random $\boldsymbol{r} \in \{0,1\}^n$. Then, for every $j \in [m]$, they use the DPF protocol to generate the secret-sharing of the unit-vector with special position $\boldsymbol{r}[S_j]$ and non-zero value equal to 1. The latter can be easily converted in a secret-sharing of the truth-table of $f_j^{\boldsymbol{r}}$: suppose that the secret-shared unit-vector $[\![\boldsymbol{v_j}]\!]$, the parties just need to output the truth table of the function

$$\sum_{\boldsymbol{s} \in \{0,1\}^n} f_j^{\boldsymbol{s}} \cdot [\![\boldsymbol{v_j}[\boldsymbol{s}]]\!]$$

The issue of this approach is that, in order to implement $\mathcal{F}_{\mathsf{Prep}}$, we would need $\sum_{\ell \in [0..B]} \hat{w}_\ell \sim s/\log\log s$ executions of the DPF. Each of these executions requires at least $\mathsf{poly}(\lambda)$ communication. The total communication would therefore be roughly $s/\log\log s \cdot \mathsf{poly}(\lambda) = \Omega(s)$ (indeed, since $s = \mathsf{poly}(\lambda)$, $\log\log s = O(\log\log\lambda)$).

THE PREPROCESSING FUNCTIONALITY $\mathcal{F}_{\mathsf{Prep}}$

Let d denote the depth of C. Split C into $\lceil d/u \rceil$ blocks, where $u := \frac{1}{2}\log\log s$ [a], as in [Cou19, Section 4.1]: let L_i denote the i-th layer of the circuit C and let w_i be its width. We consider a value $t \in [0..u-1]$ such that $\sum_{j=0}^{B} w_{j\cdot u+t} \leq \lceil s/u \rceil$ where B is $\max\{j | j \cdot u + t \leq d\}$. Such t exists [Cou19]. For every such j, let $\hat{L}_j$ denote layer $L_{j\cdot u+t}$, let $\hat{w}_j$ denote its width. For every $\ell \in [0..B]$ and $j \in [\hat{w}_\ell]$, let $C_{\ell,j}$ be the minimal subcircuit of C that computes the j-th gate of $\hat{L}_\ell$ given only values of input gates and gates in layer $\hat{L}_{\ell-1}$. Let $S_{\ell,j}$ denote the subset of gates that are input in $C_{\ell,j}$. Let $S_\ell := \bigcup_j S_{\ell,j}$, $k_\ell := |S_\ell|$ and $k_{\ell,j} := |S_{\ell,j}|$.

Initialisation: On input Init from all parties, the functionality activates. Let ι be the index of a fixed honest party.

One-Time-Truth-Table: On input $(\mathsf{OTTT}, \mathsf{id})$ from all parties, the functionality performs the following operations for every $\ell \in [0..B]$:

1. Receive $(\boldsymbol{r_i}, \boldsymbol{T_i^1}, \ldots, \boldsymbol{T_i^{\hat{w}_\ell}})$ for every corrupted party $\mathcal{P}_i$ from the adversary.
2. For every honest $\mathcal{P}_i$, sample $\boldsymbol{r_i} \xleftarrow{\$} \{0,1\}^{k_\ell}$
3. $\boldsymbol{r} \leftarrow \bigoplus_{i=1}^{N} \boldsymbol{r_i}$
4. $\forall j \in [\hat{w}_\ell] : \boldsymbol{T^j} \leftarrow \big(C_{\ell,j}\big(0 \oplus \boldsymbol{r}[S_{\ell,j}]\big), C_{\ell,j}\big(1 \oplus \boldsymbol{r}[S_{\ell,j}]\big), \ldots, C_{\ell,j}\big((2^{k_{\ell,j}} - 1) \oplus \boldsymbol{r}[S_{\ell,j}]\big)\big)$
5. For every honest $\mathcal{P}_i$, $i \neq \iota$ and $j \in [\hat{w}_\ell] : \boldsymbol{T_i^j} \xleftarrow{\$} \{0,1\}^{2^{k_{\ell,j}}}$
6. $\forall \ell \in [\hat{w}_i] : \boldsymbol{T_\iota^j} \leftarrow \boldsymbol{T^j} \oplus \bigoplus_{i \neq \iota} \boldsymbol{T_i^j}$
7. Output $(\boldsymbol{r_i}, \boldsymbol{T_i^1}, \ldots, \boldsymbol{T_i^{\hat{w}_\ell}})$ to every party $\mathcal{P}_i$.

[a] In order for our techniques to work, we need $u := c \cdot \log\log s$ for any constant $0 < c < 1$.

Fig. 17. The Preprocessing Functionality $\mathcal{F}_{\mathsf{Prep}}$

To work around this issue, we modify the construction in Fig. 15: we use succinct HSS to simultaneously run all the DPF instances corresponding to the same block of the circuit. In particular, at the beginning, each party $\mathcal{P}_i$ will input a key $\boldsymbol{K_i}$ for a PRF in NC_1. Such PRF will be used to generate, for every $\ell \in [0..B]$, the share of the shift $\boldsymbol{r_i} \in \{0,1\}^{k_\ell}$. Notice that the strings $\boldsymbol{r_1}, \ldots, \boldsymbol{r_N}$ uniquely determine the special positions of the DPFs. In other words, since the non-zero elements are all equal to 1, the parties do not need to provide any additional standard HSS input, the only communication will consist of the hashes of the special HSS inputs. Since, for every $\ell \in [0..B]$, we are running $\hat{w}_\ell$ DPF executions simultaneously, we do not need to send $\hat{w}_\ell$ digests: we can concatenate all the $\hat{w}_\ell$ special inputs and we send a single digest. In this way the total communication of the protocol will be sublinear in s.

Theorem 14. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Input}, \mathsf{Eval})$ *be a 2-party succinct HSS and let* F *be a PRF in* NC_1*. Let* F' *be another PRF. Assume the existence of authenticated point-to-point private channels. The protocol* Π_{Prep} *(see Fig. 18) UC-implements the functionality* $\mathcal{F}_{\mathsf{Prep}}$ *(see Fig. 17) in the* $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$*-hybrid model (see Fig. 12) against static, semi-honest adversaries corrupting up to* $N-1$ *parties.*

The average communication per channel of the protocol is $O(\lambda \cdot t_0(\lambda) + B \cdot t_1(\lambda, n))$ *where* $t_0(\lambda)$ *and* $t_1(\lambda, n)$ *denote the size of the succinct HSS input messages and digests respectively.*

Proof. We start by showing correctness of the correlation. Consider the execution for any $\ell \in [0..B]$. We define $\boldsymbol{r'_j} := \bigoplus_{i \leq j} \boldsymbol{r_i}$ and $\boldsymbol{r} := \bigoplus_{i \in [N]} \boldsymbol{r_i}$. We claim that, for every $j \in [N]$ and $h \in [\hat{w}_\ell]$,

$$\sum_{i=1}^{j} \boldsymbol{s_{i,j+1}^h} = \boldsymbol{u_{r'_j[S_{\ell,h}],1}}$$

THE PREPROCESSING PROTOCOL Π_{Prep}

Let n be $(\sum_{\ell\in[0..B]}\tau_\ell)/(B+1)$ where $\tau_\ell := \sum_{h=1}^{\hat{w}_\ell} 2^{k_{\ell,h}}$. Let F be a PRF in NC_1, let F' be another PRF. For every $\ell \in [0..B]$ and identity id, let g^ℓ_{id} be the special RMS program described in Fig. 19. For every $\{i,j\} \subseteq [N]$, let $\mathcal{F}^{i,j}_{\mathsf{HSS\text{-}Setup}}$ be a copy of $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$ to which only $\mathcal{P}_i$ and $\mathcal{P}_j$ have access.

Initialisation: On input Init, each party $\mathcal{P}_i$ performs the following operations

1. for every $j \neq i$, send (Init, n) to $\mathcal{F}^{i,j}_{\mathsf{HSS\text{-}Setup}}$. Receive $(\mathsf{pk}_{i,j}, \mathsf{ek}_{i,j})$ (if $i<j$) or $(\mathsf{pk}_{i,j}, \mathsf{ek}_{j,i})$ (if $i>j$) as an answer ($\mathcal{P}_j$ will receive the other half of the key).
2. $\boldsymbol{K_i} \xleftarrow{\$} \{0,1\}^\lambda$
3. $\forall j < i : \boldsymbol{K_{i,j}} \xleftarrow{\$} \{0,1\}^\lambda$
4. $\forall j < i, t \in [\lambda] : I_{i,j}[t] \xleftarrow{\$} \mathsf{HSS.Input}(\mathsf{pk}_{i,j}, \boldsymbol{K_i}[t])$
5. Send $(I_{i,j}[1], \ldots, I_{i,j}[\lambda], \boldsymbol{K_{i,j}})$ to party $\mathcal{P}_j$ for every $j < i$
6. Receive $(I_{j,i}[1], \ldots, I_{j,i}[\lambda], \boldsymbol{K_{j,i}})$ from party $\mathcal{P}_j$ for every $j > i$.

One-Time-Truth-Table: On input $(\mathsf{OTTT}, \mathsf{id})$ from the environment, the parties perform the following operations for every $\ell \in [0..B]$.

1. each party $\mathcal{P}_i$ computes $\boldsymbol{r_i} \leftarrow F(\boldsymbol{K_i}, (\mathsf{id}, \ell))$[a]
2. $\mathcal{P}_1$ sets $\boldsymbol{s^h_{1,2}} \leftarrow \boldsymbol{u_{r_1[S_{\ell,h}],1}}$ for every $h \in [\hat{w}_\ell]$
3. for $j = 2, \ldots, N$:
   - each $\mathcal{P}_i$ such that $i < j$ computes:
     (a) $(d_{i,j}, \psi_{i,j}) \xleftarrow{\$} \mathsf{HSS.Hash}\big(\mathsf{pk}_{i,j}, 0, (\boldsymbol{s^1_{i,j}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j}})\big)$[b]
     (b) Send $d_{i,j}$ to $\mathcal{P}_j$
     (c) $(\boldsymbol{s^1_{i,j+1}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j+1}}) \leftarrow \mathsf{HSS.Eval}(\mathsf{ek}_{i,j}, g^\ell_{\mathsf{id}}, I_{j,i}[1], \ldots, I_{j,i}[\lambda], \psi_{i,j})$
   - $\mathcal{P}_j$ computes:
     (a) Receive $d_{i,j}$ from $\mathcal{P}_i$ for every $i < j$
     (b) For every $i < j$, compute

$$(\boldsymbol{s^1_{j,j+1}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{j,j+1}}) \leftarrow \sum_{i=1}^{j-1} \mathsf{HSS.Eval}(\mathsf{ek}_{j,i}, g^\ell_{\mathsf{id}}, I_{j,i}[1], \ldots, I_{j,i}[\lambda], d_{i,j})$$

4. For every $h \in [\hat{w}_\ell]$, $y \in \{0,1\}^{k_{\ell,h}}$, each party $\mathcal{P}_i$ computes

$$\boldsymbol{T^h_i}[y] \leftarrow \bigoplus_{z\in\{0,1\}^{k_{\ell,h}}} C_{\ell,h}(y \oplus z) \cdot \boldsymbol{s^h_{i,N+1}}[z] \oplus \bigoplus_{j<i} F'(K_{i,j}, (\mathsf{id}, \ell, h, y)) \oplus \bigoplus_{j>i} F'(K_{j,i}, (\mathsf{id}, \ell, h, y))$$

5. Each party $\mathcal{P}_i$ outputs $(\boldsymbol{r_i}, \boldsymbol{T^1_i}, \ldots, \boldsymbol{T^{\hat{w}_\ell}_i})$.

[a] We assume that the output of the PRF is truncated at the right length k_ℓ.
[b] If the given vector is too long, we split into chunks of n elements and we hash each of them.

Fig. 18. The Preprocessing Protocol Π_{Prep}

The Special RMS Program g^{ℓ}_{id}

For every $h \in [\hat{w}_\ell]$ and $i \in [k_\ell]$, let $\delta^{\ell}_{h,i}$ be the $k_{\ell,h}$-bit string that is zero everywhere except for the position t, where i is the t-th element in $S_{\ell,h}$ (t might also not exist).

Special Input: a vector $(\boldsymbol{s^1}, \ldots, \boldsymbol{s^{\hat{w}_\ell}})$ where, for every $h \in [\hat{w}_\ell]$, $\boldsymbol{s^h} \in \mathbb{Z}_2^{\tau_{\ell,h}}$ and $\tau_{\ell,h} = 2^{k_{\ell,h}}$.
Standard Input: a PRF key $\boldsymbol{K} \in \{0,1\}^\lambda$

1. $\boldsymbol{r} \leftarrow F(\boldsymbol{K}, (\mathsf{id}, \ell))$
2. $\forall h \in [\hat{w}_\ell] : \boldsymbol{v_0^h} \leftarrow \boldsymbol{s^h}$
3. For $i = 1, \ldots, k_\ell$:
   (a) $\forall h \in [\hat{w}_\ell], j \in \{0,1\}^{k_{\ell,h}} : \boldsymbol{v_i^h}[j] \leftarrow \boldsymbol{v_{i-1}^h}[j] + \boldsymbol{r}[i] \cdot (\boldsymbol{v_{i-1}^h}[j \oplus \delta^{\ell}_{h,i}] - \boldsymbol{v_{i-1}^h}[j])$
4. Output $\boldsymbol{v^1_{k_\ell}}, \ldots, \boldsymbol{v^{\hat{w}_\ell}_{k_\ell}}$

Fig. 19. The Special RMS Program g^{ℓ}_{id}

The claim is clearly true for $j = 1$. Now suppose that the claim is true for $j-1$. We prove it also for j. We observe that, by the correctness of HSS, we have

$$\begin{aligned}
\sum_{i=1}^{j} (\boldsymbol{s^1_{i,j+1}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j+1}}) &= \sum_{i=1}^{j-1} (\boldsymbol{s^1_{i,j+1}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j+1}}) + (\boldsymbol{s^1_{j,j+1}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{j,j+1}}) \\
&= \sum_{i=1}^{j-1} \mathsf{HSS.Eval}(\mathsf{ek}_{i,j}, g^{\ell}_{\mathsf{id}}, I_{j,i}[1], \ldots, I_{j,i}[\lambda], \psi_{i,j}) \\
&\quad + \sum_{i=1}^{j-1} \mathsf{HSS.Eval}(\mathsf{ek}_{j,i}, g^{\ell}_{\mathsf{id}}, I_{j,i}[1], \ldots, I_{j,i}[\lambda], d_{i,j}) \\
&= \sum_{i=1}^{j-1} g^{\ell}_{\mathsf{id}}(\boldsymbol{s^1_{i,j}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j}}, \boldsymbol{K_j})
\end{aligned}$$

We also observe that g^{ℓ}_{id} is linear in $\boldsymbol{s^1_{i,j}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j}}$ (it only permutes the entries of the vectors). So,

$$\begin{aligned}
\sum_{i=1}^{j} \boldsymbol{s^h_{i,j+1}} &= \sum_{i=1}^{j-1} g^{\ell}_{\mathsf{id}}(\boldsymbol{s^1_{i,j}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j}}, \boldsymbol{K_j}) \\
&= g^{\ell}_{\mathsf{id}} \left(\sum_{i=1}^{j-1} (\boldsymbol{s^1_{i,j}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j}}), \boldsymbol{K_j} \right) \\
&= g^{\ell}_{\mathsf{id}} \left(\boldsymbol{u_{r'_{j-1}[S_{\ell,1}],1}}, \ldots, \boldsymbol{u_{r'_{j-1}[S_{\ell,\hat{w}_\ell}],1}}, \boldsymbol{K_j} \right) \\
&= (\boldsymbol{u_{r'_j[S_{\ell,1}],1}}, \ldots, \boldsymbol{u_{r'_j[S_{\ell,\hat{w}_\ell}],1}}).
\end{aligned}$$

We conclude that

$$\sum_{i=1}^{N} \boldsymbol{s^h_{i,N+1}} = \boldsymbol{u_{r[S_{\ell,h}],1}}$$

Now, pick any $h \in [\hat{w}_\ell]$ and $y \in \{0,1\}^{k_{\ell,h}}$, we observe that, for every $h \in [\hat{w}_\ell]$,

$$\begin{aligned}
\bigoplus_{i\in[N]} \boldsymbol{T_i^h}[y] &= \bigoplus_{i\in[N]} \Big(\bigoplus_{z\in\{0,1\}^{k_{\ell,h}}} C_{\ell,h}(y\oplus z)\cdot \boldsymbol{s^h_{i,N+1}}[z] \\
&\qquad \oplus \bigoplus_{j<i} F'(K_{i,j},(\mathsf{id},\ell,h,y)) \\
&\qquad \oplus \bigoplus_{j>i} F'(K_{j,i},(\mathsf{id},\ell,h,y))\Big) = \\
&= \bigoplus_{z\in\{0,1\}^{k_{\ell,h}}} C_{\ell,h}(y\oplus z)\cdot \left(\bigoplus_{i\in[N]} \boldsymbol{s^h_{i,N+1}}[z]\right) \\
&= \bigoplus_{z\in\{0,1\}^{k_{\ell,h}}} C_{\ell,h}(y\oplus z)\cdot \boldsymbol{u_{r[S_{\ell,h}],1}}[z] \\
&= C_{\ell,h}(y\oplus \boldsymbol{r}[S_{\ell,h}]).
\end{aligned}$$

We now focus our attention on security. We consider the simulator $\mathcal{S}$ that just runs the protocol using dummy inputs for the honest parties and, for every $\ell \in [0..B]$, sends $(\boldsymbol{r_i}, \boldsymbol{T_i^1}, \ldots, \boldsymbol{T}_i^{\hat{w}_\ell})$ to $\mathcal{F}_{\mathsf{Prep}}$ for every corrupted party $\mathcal{P}_i$. We show that no semi-honest adversary can distinguish between real world and ideal world using a sequence of indistinguishable hybrids.

Hybrid 0. This hybrid corresponds to the real world.

Hybrid 1. In this hybrid, we fix an honest party $\mathcal{P}_\iota$. For every $\ell \in [0..B]$, we compute the output of $\mathcal{P}_\iota$ as follows

1. $\forall i \in [N] : \boldsymbol{r_i} \xleftarrow{\$} F(\boldsymbol{K_i}, (\mathsf{id}, \ell))$
2. $\boldsymbol{r} \leftarrow \bigoplus_{i\in[N]} \boldsymbol{r_i}$
3. $\forall h \in [\hat{w}_\ell], y \in \{0,1\}^{k_{\ell,h}} : \boldsymbol{T_\iota^h}[y] \leftarrow C_{\ell,h}(y \oplus \boldsymbol{r}[S_{\ell,h}]) \oplus \bigoplus_{i\neq\iota} \boldsymbol{T_i^h}[y]$

By the correctness of the protocol, this hybrid is indistinguishable from the previous one.

Hybrid 2. In this hybrid, for every honest party $\mathcal{P}_i$ with $i \neq \iota$, every $\ell \in [0..B]$ and every $h \in [\hat{w}_\ell]$, we sample $\boldsymbol{T_i^h} \xleftarrow{\$} \mathbb{Z}_2^{\tau_{\ell,h}}$. This hybrid is indistinguishable from the previous one due to the security of the PRF in F'.

Hybrid 3. In this hybrid, for every honest party $\mathcal{P}_i$, during the initialisation, instead of inputting $\boldsymbol{K_i}$ in the succinct HSS scheme, we input 0. This hybrid is indistinguishable from the previous one due to the input privacy of succinct HSS.

Hybrid 4. In this hybrid, for every honest party $\mathcal{P}_i$ and every $\ell \in [0..B]$, instead of hashing $(\boldsymbol{s^1_{i,j}}, \ldots, \boldsymbol{s^{\hat{w}_\ell}_{i,j}})$ for every $j > i$, we hash $\mathbf{0}$. This hybrid is indistinguishable from the previous one thanks to the hasher privacy of the succinct HSS scheme.

Hybrid 5. This hybrid corresponds to the ideal world. For every honest party $\mathcal{P}_i$ and every $\ell \in [0..B]$, we do not evaluate F on input $\boldsymbol{K_i}$ and (id, ℓ) anymore. We instead sample the outputs $\boldsymbol{r_i}$ at random. This hybrid is indistinguishable from the previous one due to the security of the PRF in NC_1.

This ends the proof. □

Putting together Corollary 4, Theorem 13 and Theorem 14, we obtain the following corollary.

Corollary 5. *Assume that one of the following holds:*

- *DCR over the Paillier group,*
- *DDH, hidden subgroup assumption, small exponent assumption, the uniformity assumption and $n^{1/3}$-ary EDDH over class groups,*
- *LWE with superpolynomial modulus-to-noise ratio.*

For any N-party functionality $\mathcal{F}$ represented by a layered boolean circuit C of size s with k inputs and m outputs, there exists a protocol that realises $\mathcal{F}$ against semi-honest adversaries corrupting up to $N-1$ parties in the plain model with total communication.

$$O(k + N \cdot (m + s/\log\log s)) + O(N^2 \cdot s^{2/3\cdot(1+\epsilon)}) \cdot \mathsf{poly}(\lambda)$$

for any constant $\epsilon > 0$.

Proof. We build our MPC protocol by considering the protocol of [Cou19] (see Theorem 13) and instantiating $\mathcal{F}_{\mathsf{Prep}}$ using Π_{Prep}.

Under any of the assumptions of Corollary 5, it is possible to build succinct HSS schemes with hash input bound n, where the input message size is $\mathsf{poly}(\lambda)$ and independent of n. The key size and the digest size is instead $O(n^{2/3}) \cdot \mathsf{poly}(\lambda)$. Using any of these schemes, the communication of Π_{Prep} (averaged over all channels) is $O(n^{2/3}) \cdot B \cdot \mathsf{poly}(\lambda)$.

To this, we need to add the communication complexity of a protocol that implements the trusted setup. Since all the considered succinct HSS schemes satisfy setup sublinearity, the communication complexity of the setup is $O(N^2 \cdot n^{2/3}) \cdot \mathsf{poly}(\lambda)$.

We observe that $\sum_{\ell\in[0..B]} \hat{w}_\ell \leq \lceil 2s/\log\log s\rceil$ and

$$n \leq \sum_{\ell\in[0..B]} \tau_\ell/B \leq \lceil 2s/\log\log s\rceil \cdot 2^{2^u}/B = O(s/\log\log s \cdot 2^{\sqrt{\log s}})/B.$$

This is because, since each $C_{i,j}$ is a 2-ary circuit of depth at most $u = \frac{1}{2}\log\log s$, the number of inputs of $C_{i,j}$ is at most $2^u = \sqrt{\log s}$. That proves that $n = o(s^{1+\epsilon})/B$ for any constant $\epsilon > 0$. We conclude that the total communication of the protocol is therefore bounded by

$$O(k + N \cdot (m + s/\log\log s)) + O(N^2 \cdot s^{2/3\cdot(1+\epsilon)}) \cdot \mathsf{poly}(\lambda).$$

□

9 Asymptotically Better Sublinear Communication MPC for Layered Circuits

The N-party protocol for layered circuits we presented in the previous section achieves $O(s/\log\log s)$ communication, where s denotes the size of the circuit we want to evaluate. While this is an interesting result, as it proves that sublinear communication MPC without FHE is possible for every number of parties, the $\log\log s$ factor is quite small. In this section, we show that using succinct HSS, it is possible to build N-party protocols that achieve $O(s/\log s)$ communication for a slightly smaller class of circuit.

Before explaining our idea, we introduce the notion of layered RMS program. Essentially, this consists of an RMS program in which the gates are partitioned into layers $S_1, \ldots, S_L$. If a gate belongs to the layer S_i, all its input must belong to the above layers $\bigcup_{j<i} S_j$. We require that S_1 consists of all and only the input gates. On the other hand, S_L consists of all and only the output gates.

Definition 27 (Layered RMS Program). *A layered RMS program over the ring R consists of a direct graph (D, E) where the nodes are partitioned into L layers $S_1, \ldots, S_L$ for some $L \in \mathbb{N}$. Each node belongs to exactly one of the following types:*

- ***Input Gate.*** *It has in-degree 0 and it belongs to S_1.*
- ***Linear Gate.*** *It has unbounded in-degree. If the gate belongs to the layer S_j, the parent nodes must belong to $\bigcup_{i<j} S_i$. Each of these gates is associated with a vector $\boldsymbol{z}$ in R^m where m is the in-degree of the node.*
- ***Multiplication Gate.*** *It has in-degree 2. If the gate belongs to the layer S_j, one parent node, called the* input parent*, must belong to S_1, the other one, called the* internal parent*, must instead belong to $\bigcup_{i<j} S_i$.*

– ***Output Gate.*** *It is a node with in-degree 1 and out-degree 0. The gate must belong to the layer S_L, the parent node must belong to $\bigcup_{i<L} S_j$.*

We use $k := |S_1|$ to denote the number of input gates and $s := \sum_{j=1}^{L} |S_j|$ to denote the total number of gates. For every $j \in [L]$, we use U_j to denote the list of all input parents of multiplication gates in layer S_j. Similarly, we use V_j to denote the list of all internal parents of multiplication gates in layer S_j.

We say that a family of layered RMS programs $(C_n)_{n\in\mathbb{N}}$, where C_n has $k(n)$ input gates and $s(n)$ gates, is efficient if there exists a polynomial p such that $s(n) \leq p(k(n))$ except for a finite number of $n \in \mathbb{N}$.

Below, we formalise how a layered RMS program is evaluated. The procedure is analogous to the evaluation of a standard RMS program.

Definition 28 (Evaluation of Layered RMS Programs). *Let C be a layered RMS program over R with k input gates. Let $\boldsymbol{x}$ be a vector of length k over R. The evaluation of C over x consists of the unique labelling of the nodes in C satisfying the following properties:*

- *The i-th input gate of C is labelled by x_i*
- *Every linear gate is labelled by the inner-product $\langle \boldsymbol{z}, \boldsymbol{x} \rangle$ where $\boldsymbol{z}$ is the vector associated with the gate and $\boldsymbol{x}$ is the vector consisting of the labels of the parent nodes.*
- *Every multiplication gate is labelled by $x \cdot y$ where x and y are the labels associated with the parent nodes.*
- *Every output gate is associated with the same label of the parent node.*

The result of the evaluation consists of the list of labels of the output nodes.

Now that we have formalised the necessary notions, we presents an N-party protocol Π_{RMS} (see Fig. 21) for the secure evaluation of layered RMS programs. This will be the basic building block for our sublinear communication MPC protocol. The communication complexity of Π_{RMS} will scale linearly in the number of inputs, outputs and depth of the program. It will instead be sublinear in the width.

The protocol relies on a setup that generates and distributes a pair of evaluation keys for a 2-party succinct HSS scheme to each pair of participants (as we did in our N-party DPF). After that, the evaluation of the RMS programs begins from the parties providing their inputs: in order to supply a value x, party $\mathcal{P}_i$ will insert it in all 2-party succinct HSS executions it is involved.

After this phase, the protocol will proceed by evaluating the RMS program layer after layer. Throughout its execution, we will satisfy and invariant: after processing the ℓ-th layer, the parties will hold an additive secret-sharing of the content of all wires in the ℓ-th layer. Before the protocol begins, this is of course true: the share of a party providing the input x will be x, the shares of the other parties will all be 0.

Now, suppose that the parties hold an additive secret-sharing of all values in the ℓ-th layer. Using local computations, they can easily derive additive secret-sharings of all outputs of linear gates in the $(\ell+1)$-th layer. The hard part is the evaluation of the multiplication gates. Luckily, since we are dealing with an RMS program, we know that one of the inputs of every multiplication gate is an input to the whole protocol. The value of the other input will instead be secret-shared among all parties. In order to evaluate the multiplications in the $(\ell+1)$-th layer, for every $i \in [N]$, each party $\mathcal{P}_j$ will collect the shares of all wires that need to be multiplied by inputs that were provided by $\mathcal{P}_i$. Then, it will hash them using the succinct HSS instance with party $\mathcal{P}_i$ and will send the digest to $\mathcal{P}_i$. At that point, the two parties $\mathcal{P}_i$ and $\mathcal{P}_j$ will run the special RMS program that multiplies the hashed shares by the corresponding inputs of party $\mathcal{P}_i$.

Concretely, suppose that, in the $(\ell+1)$-th layer, we need to multiply the input x supplied by $\mathcal{P}_i$ by the secret-shared internal wire $[\![y]\!]$. For every $j \in [N]$, let y_j be the share of party $\mathcal{P}_j$. We observe that, in our protocol, y_j is hashed along with other values by $\mathcal{P}_j$ and the resulting digest is sent to $\mathcal{P}_i$. After the evaluation of the special RMS program, $\mathcal{P}_i$ and $\mathcal{P}_j$ obtain respectively shares $z_{i,j}$ and z_j such that $z_{i,j} + z_j = x \cdot y_j$. We therefore notice that $z_1, \ldots, z_N$, where

$$z_i := x \cdot y_i + \sum_{j \neq i} z_{i,j}$$

is an additive secret-sharing of $x \cdot y$. So, to derive the secret-sharing of the outputs of the multiplication gates, party $\mathcal{P}_i$ needs to add all the shares obtained from the succinct HSS evaluations (there is one with

every party $\mathcal{P}_j$) and then correct the result by adding the product of the inputs it provided by its shares of the internal wires.

By performing these operations for every layer, the parties will obtain a secret-sharing of the output.

THE FUNCTIONALITY $\mathcal{F}_{\mathsf{RMS}}$

Initialisation: The functionality activates after receiving (Init, n) from all parties. Let ι be the index of an honest party.

Input: On input $(\mathsf{Input}, \mathsf{id}, i)$ from all parties. Receive $x \in \mathbb{Z}_q$ from P_i and store it under the identity id.

RMS: On input $(\mathsf{RMS}, C, \mathsf{id}_1, \ldots, \mathsf{id}_k)$ for some $k \in \mathbb{Z}$ from each party $\mathcal{P}_i$, where C is the description of a layered RMS program over $\mathbb{Z}_q$ with k inputs, m outputs and with at most n multiplications per layer, the functionality outputs $\boldsymbol{y} \leftarrow C(x_1, \ldots, x_k)$ to all parties, where, for every $j \in [k]$, x_j denotes the element stored under the identity id_j.

Fig. 20. The Functionality $\mathcal{F}_{\mathsf{RMS}}$

Theorem 15. *Assume the existence of 2-party succinct HSS over $\mathbb{Z}_2$ and PRFs in NC_1. Suppose that the parties are connected by authenticated point-to-point channels. Then, the protocol in Fig. 21 UC-implements the functionality $\mathcal{F}_{\mathsf{RMS}}$ (see Fig. 20) against a semi-honest, static adversary corrupting up to $N-1$ parties in the $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$-hybrid model (see Fig. 12).*

The total communication complexity required in order to evaluate a layered RMS program with k inputs, L layers and m outputs is

$$N \cdot k \cdot t_0(\lambda, n) + N \cdot m + N^2 \cdot L \cdot t_1(\lambda, n),$$

where $t_0(\lambda, n)$ and $t_1(\lambda, n)$ denote the input message size and the digest size of the succinct HSS scheme. The round complexity is $\Theta(L)$.

Proof. We start by showing correctness. We proceed by induction over the layers to show that $\sum_{i\in[N]} \boldsymbol{v_{\ell,i}}$ corresponds to the correct evaluation of the ℓ-th layered of the program. It is easy to see that $\sum_{i\in[N]} \boldsymbol{v_{1,i}} = \boldsymbol{x}$ where $\boldsymbol{x}$ denotes the vector of inputs. It remains to show that, if the hypothesis holds for the first $\ell-1$ layers, so does for the ℓ-th layer. The claim comes for free for linear gates. We therefore focus on multiplication gates. Consider a gate and suppose that the input parent is provided by party $\mathcal{P}_i$. Let u_i be its value. Let v_j be the value of the entry of $\boldsymbol{v_{1,j}}, \ldots, \boldsymbol{v_{\ell-1,j}}$ corresponding to the internal parent. Observe that by inductive hypothesis, in the evaluation of the RMS program, the value of the internal parent is $\sum_{j\in[N]} v_j$. At the end of their succinct HSS execution, $\mathcal{P}_i$ and $\mathcal{P}_j$ obtain a two-party secret-sharing of $u_i \cdot v_j$. For every $j \neq i$, the entry of $\boldsymbol{v_{\ell,j}}$ corresponding to the considered multiplication gate will be the succinct HSS share obtained by $\mathcal{P}_j$ in the above procedure. For party $\mathcal{P}_i$, instead, the entry will be the addition between $u_i \cdot v_i$ and the sum of all the succinct HSS shares obtained by $\mathcal{P}_i$ (observe that $\mathcal{P}_i$ is involved in $N-1$ executions of the procedure, each one with a different $\mathcal{P}_j$). We conclude by observing that the sum of the resulting shares is $\sum_{j\in[N]} u_i \cdot v_j = u_i \cdot v$. This is exactly what we wanted.

This ends the proof of correctness.

Security is a straightforward application of the hasher privacy, the input privacy, and the pseudorandom shares of the succinct HSS scheme. After receiving the inputs of the corrupted parties, the simulator will just forward them to the functionality and run the protocol using dummy inputs for the honest players. It will then provide the functionality with the output shares of the corrupted players. □

We finally explain how to use the N-party protocol for RMS protocols to build N-party computation protocols with sublinear communication in the circuit size. Before describing the details, we however recall the following result, which shows that every depth d circuit can be converted into an RMS program of depth $O(2^d)$.

Theorem 16 ([Weg00, BGI16]). *If C is a layered boolean circuit of depth d and width w, then, C is computable by a layered RMS program of depth $O(2^d)$ with at most w multiplications in each layer.*

SUBLINEAR COMMUNICATION MPC FOR RMS PROGRAMS Π_{RMS}

Let N be the number of parties. For every $\{i,j\} \subseteq [N]$, let $\mathcal{F}^{i,j}_{\mathsf{HSS\text{-}Setup}}$ be a copy of $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$ to which only $\mathcal{P}_i$ and $\mathcal{P}_j$ have access.

Initialisation: On input (Init, n), each party $\mathcal{P}_i$ performs the following operations:

1. for every $j \neq i$, send (Init, n) to $\mathcal{F}^{i,j}_{\mathsf{HSS\text{-}Setup}}$. Receive $(\mathsf{pk}_{i,j}, \mathsf{ek}_{i,j})$ (if $i < j$) or $(\mathsf{pk}_{i,j}, \mathsf{ek}_{j,i})$ (if $i > j$) as an answer ($\mathcal{P}_j$ will obtain the other half of the key).

Input: In order to input a value $x \in \mathbb{Z}_q$ under the identity id, party $\mathcal{P}_i$ performs the following operations:

1. $\forall j < i : I_{i,j} \xleftarrow{\$} \mathsf{HSS.Input}(\mathsf{pk}_{j,i}, x)$
2. $\forall j > i : I_{i,j} \xleftarrow{\$} \mathsf{HSS.Input}(\mathsf{pk}_{i,j}, x)$
3. Send $I_{i,j}$ to P_j for every $j \neq \iota$.
4. Store $(I_{i,j})_{j \neq i}$ and x under id

Each other party $\mathcal{P}_j$ stores $I_{i,j}$ under the identity id.

RMS: Given identities $\mathsf{id}_1, \ldots, \mathsf{id}_k$ and an RMS program C with k inputs, L layers $S_1, \ldots, S_L$ and with at most n multiplications per layer, each party $\mathcal{P}_i$ computes the following operations:

1. Each party retrieves the values stored under the identities $\mathsf{id}_1, \ldots, \mathsf{id}_k$
2. Define a vector $\boldsymbol{v_{1,i}}$ with the same dimension as the number of gates in S_1. Each entry is assigned with the value of the corresponding input wire if the latter was provided by $\mathcal{P}_i$, with 0 otherwise.
3. Evaluate the gates in S_ℓ for $\ell = 2, \ldots, L$ as follows:
   (a) For every $j \in [N] \setminus \{i\}$, using $\mathsf{pk}_{i,j}$ or $\mathsf{pk}_{j,i}$ (depending on whether $j < i$ or $j > i$), hash all entries in $\boldsymbol{v_{1,i}}, \ldots, \boldsymbol{v_{\ell-1,i}}$ corresponding to gates in V_ℓ where the associated input parent was provided by $\mathcal{P}_j$. Send the resulting hash $d^\ell_{i,j}$ to $\mathcal{P}_j$ and store the corresponding private information $\psi^\ell_{i,j}$.
   (b) Receive $d^\ell_{j,i}$ from every party $\mathcal{P}_j$.
   (c) Evaluate all linear gates in S_ℓ on $\boldsymbol{v_{1,i}}, \ldots, \boldsymbol{v_{\ell-1,i}}$.
   (d) For every multiplication gate in S_ℓ where the value of the input parent is provided by another party $\mathcal{P}_j$, use $\psi^\ell_{i,j}$ and $I_{j,i}$ to evaluate the special RMS program (see Lemma 1) that multiplies the input parent by the share of the internal parent hashed in $d^\ell_{i,j}$.
   (e) For every multiplication gate in S_ℓ where the value of the input parent is provided by $\mathcal{P}_i$, for every $j \neq i$, use $d^\ell_{j,i}$ and $I_{i,j}$ to evaluate the special RMS program (see Lemma 1) that multiplies the input parent by the share of the internal parent hashed in $d^\ell_{j,i}$. Finally, add to the resulting HSS shares the multiplication between the value of the input parent and the entry in $\boldsymbol{v_{1,i}}, \ldots, \boldsymbol{v_{\ell-1,i}}$ corresponding to the internal parent.
   (f) Store the values obtained in steps 3c, 3d and 3e into a vector $\boldsymbol{v_{\ell,i}}$.
4. Broadcast $\boldsymbol{v_{L,i}}$ and receive $(\boldsymbol{v_{L,j}})_{j \neq i}$.
5. Output $\sum_{j \in [N]} \boldsymbol{v_{L,j}}$.

Fig. 21. Sublinear Communication MPC for RMS Programs Π_{RMS}

To construct a sublinear communication MPC protocol for a layered circuit C, we follow the idea of [BGI16]: we split the layers of C into blocks $B_1, \ldots, B_L$ of $O(\log \lambda)$ depth. Thanks to Theorem 16, each of these blocks can be converted into a $\mathsf{poly}(\lambda)$-sized RMS program which can be evaluated using Π_{RMS} (see Fig. 21). At the end of the execution of Π_{RMS} on the ℓ-th block, the players will end up with an additive secret-sharing of the outputs of B_ℓ. Such shares would need to be reinput into Π_{RMS} for the evaluation of $B_{\ell+1}$. If we perform this in a naive way, however, that operation would require $N^2 \cdot k_{\ell+1} \cdot \mathsf{poly}(\lambda)$ communication where $k_{\ell+1}$ denotes the input length of $B_{\ell+1}$. To avoid this $N \cdot \mathsf{poly}(\lambda)$ multiplicative overhead, we adopt the following trick: we let each party $\mathcal{P}_i$ select a key for a PRF in NC_1. Such key will be input in Π_{RMS} at the very beginning of the sublinear communication MPC protocol. Instead of directly inputting the shares of the results produced by B_ℓ into Π_{RMS}, we let each party encrypt their shares under their PRF key, broadcasting the ciphertext. We can then use the PRF keys already input in Π_{RMS} to recover the actual output of B_ℓ inside Π_{RMS}. In this way, the operation requires only $N \cdot k_\ell$ communication.

THE FUNCTIONALITY $\mathcal{F}_{\mathsf{MPC}}$

MPC: On input (C, x_i) from each party $\mathcal{P}_i$ where C is a layered circuit and x_i is the input of party $\mathcal{P}_i$, the functionality outputs $\boldsymbol{y} \leftarrow C(x_1, \ldots, x_N)$ to all the parties.

Fig. 22. The Functionality $\mathcal{F}_{\mathsf{MPC}}$

Theorem 17. *Assume the existence of point to point channels and PRFs in NC_1. The protocol in Fig. 23 UC-realises the functionality $\mathcal{F}_{\mathsf{MPC}}$ in Fig. 22 in the $\mathcal{F}_{\mathsf{RMS}}$-hybrid model against a semi-honest, static adversary corrupting up to $N-1$ parties.*

If $\mathcal{F}_{\mathsf{RMS}}$ is implemented using the protocol in Fig. 21, the total communication is

$$N \cdot \sum_{\ell=1}^{L} k_\ell + N^2 \cdot \lambda \cdot t_0(\lambda, n) + N^2 \cdot \sum_{\ell=1}^{L} 2^{d_\ell} \cdot t_1(\lambda, n) + N \cdot m.$$

where m is the output size of C, k is the input size and $t_0(\lambda, n)$ and $t_1(\lambda, n)$ denote the input message size and the digest size of the succinct HSS scheme. The round complexity is $\Theta(\sum_{\ell=1}^{L} 2^{d_\ell})$.

SUBLINEAR COMMUNICATION MPC

Let C be a layered circuit over $\mathbb{Z}_q$ of width n. We split C into L blocks $B_1, \ldots, B_L$ of consecutive layers. Let d_ℓ be the depth of B_ℓ. We require that $d_\ell = O(\log \lambda)$. Let k_ℓ denote the input length of B_ℓ. Let F be a PRF in NC_1.

MPC: Let sid be the session identity, let $\boldsymbol{x}$ be the vector of inputs. Each party $\mathcal{P}_i$ performs the following operations.

1. Send (Init, n) to $\mathcal{F}_{\mathsf{RMS}}$
2. $\boldsymbol{K_i} \xleftarrow{\$} \{0,1\}^\lambda$
3. Send $(\mathsf{Input}, j, (\mathsf{sid}, j, t))$ to $\mathcal{F}_{\mathsf{RMS}}$ for every $j \in [N]$ and $t \in [\lambda]$. If $j = i$, provide $\mathcal{F}_{\mathsf{RMS}}$ with $\boldsymbol{K_i}[t]$.
4. For each input wire t of C whose value is provided by $\mathcal{P}_i$, $\mathcal{P}_i$ broadcasts $\boldsymbol{c_1}[t] \leftarrow \boldsymbol{x}[t] \oplus F(\boldsymbol{K_i}, (1, t))$.
5. For each other input wire t, receive $\boldsymbol{c_1}[t]$ from the party providing the corresponding value.
6. Let $\boldsymbol{c_1}$ be the vector $(\boldsymbol{c_1}[1], \ldots, \boldsymbol{c_1}[k])$.
7. For $\ell = 1, \ldots, L$, call $\mathcal{F}_{\mathsf{RMS}}$ on input $(\mathsf{RMS}, \widetilde{B}_\ell^{\boldsymbol{c_\ell}}, (\mathsf{sid}, j, t)_{j \in [N], t \in [\lambda]})$ and receive $\boldsymbol{c_{\ell+1}}$ as output. (see Fig. 24)
8. Output $\boldsymbol{c_{L+1}}$.

Fig. 23. Sublinear Communication MPC for Layered Programs

Putting the results of Theorem 17 and Theorem 15 together, we obtain the following corollary.

THE RMS PROGRAM $\widetilde{B}_\ell^{\boldsymbol{c_\ell}}$

Input: Keys $\boldsymbol{K_1}, \ldots, \boldsymbol{K_N} \in \{0,1\}^\lambda$

1. If $\ell = 1$, for every $t \in [k_\ell]$, compute $x_t \leftarrow \boldsymbol{c_\ell}[t] \oplus F(\boldsymbol{K_j}, (\ell, t))$ where $\mathcal{P}_j$ is the party providing the t-th input.
2. If $\ell > 1$, for every $t \in [k_\ell]$, compute $\boldsymbol{x}[t] \leftarrow \boldsymbol{c_\ell}[t] \oplus \bigoplus_{j\in[N]} F(\boldsymbol{K_j}, (\ell, t))$
3. $\boldsymbol{y} \leftarrow B_\ell(\boldsymbol{x})$
4. If $\ell = L$, output $\boldsymbol{y}$.
5. If $\ell < L$, for every $t \in [k_{\ell+1}]$, compute $\boldsymbol{c_{\ell+1}}[t] \leftarrow \boldsymbol{y}[t] \oplus \bigoplus_{j\in[N]} F(\boldsymbol{K_j}, (\ell+1, t))$
6. Output $\boldsymbol{c_{\ell+1}}$.

Fig. 24. The RMS Program $\widetilde{B}_\ell^{\boldsymbol{c_\ell}}$

Corollary 6. *Assume the existence of a succinct HSS scheme with tight setup and PRFs in* NC_1*. Let* $t_0(\lambda, n)$*,* $t_1(\lambda, n)$*,* $t_2(\lambda, n)$ *be the message input size, the digest size and the key size of the succinct HSS keys respectively.*

For any constant c *and every* N*-party functionality* $\mathcal{F}$ *represented by a layered boolean circuit* C *of size* s*, depth* d*, width* n*, with* k *inputs and* m *outputs, there exists a protocol that realises* $\mathcal{F}$ *against semi-honest adversaries corrupting up to* $N-1$ *parties in the plain model with total communication.*

$$O(N \cdot (k + m + s/\log s)) + N^2 \cdot \lambda \cdot t_0(\lambda, n) \\ + O(N^2 \cdot (d/\log s) \cdot s^{1/c}) \cdot t_1(\lambda, n) + N^2 \cdot t_2(\lambda, n) \cdot \mathsf{poly}(\lambda)$$

The round complexity is $\Omega(d/\log s \cdot s^{1/c})$*.*

Proof. We consider the composition of the protocols of Theorem 17 and Theorem 15. We observe that they rely on N^2 executions of the trusted setup $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$. Since the succinct HSS scheme has a tight setup, the latter can be implemented with $t_2(\lambda, n) \cdot \mathsf{poly}(\lambda)$ communication.

Let $S_1, \ldots, S_d$ be the layers of C. Let $u := \frac{1}{c} \cdot \log s$. We consider $t \in [0..u-1]$ such that $\sum_{j=0}^{a_t} |S_{u\cdot j+t}| \leq \lceil s/u \rceil$, where $a_t := \max\{j | u \cdot j + t \leq d\} = O(d/u)$. Such t exists, otherwise we would have

$$|C| = \sum_{t=0}^{u-1} \sum_{j=0}^{a_t} |S_{u\cdot j+t}| > u \cdot \lceil s/u \rceil \geq s.$$

We define the block B_1 as the one consisting of the layers $S_1, \ldots, S_t$. For every $j \in [a_t - 1]$, we define B_j as the block consisting of the layers $S_{(j-1)\cdot u+t}, \ldots, S_{j\cdot u+t}$. We define the block B_{a_t} as the one consisting of the layers $S_{a_t\cdot u+t}, \ldots, S_d$. Notice that the depth d_ℓ of each block B_ℓ is at most u. If we run the protocol in Fig. 23 with such choice of blocks, the communication complexity becomes

$$O(N \cdot (k + m + s/u)) + N^2 \cdot \lambda \cdot t_0(\lambda, n) \\ + O(N^2 \cdot (d/u) \cdot 2^u) \cdot t_1(\lambda, n) + N^2 \cdot t_2(\lambda, n) \cdot \mathsf{poly}(\lambda).$$

□

The following result shows the communication complexity of our MPC protocol when the succinct HSS scheme is instantiated with the constructions of Section 4.

Corollary 7. *Assume that one of the following holds:*

- *DCR over the Paillier group,*
- *DDH, hidden subgroup assumption, small exponent assumption, the uniformity assumption and* $n^{1/3}$*-ary EDDH over class groups,*
- *LWE with superpolynomial modulus-to-noise ratio.*

For any constant $c > 0$ and for every N-party functionality $\mathcal{F}$ represented by a layered boolean circuit C of size s, depth d, width n, with k inputs and m outputs, there exists a protocol that realises $\mathcal{F}$ against semi-honest adversaries corrupting up to $N-1$ parties in the plain model with total communication.

$$O(N \cdot (k + m + s/\log s)) + N^2 \cdot (d/\log s) \cdot s^{1/c} \cdot \mathsf{poly}(\lambda) + N^2 \cdot n^2 \cdot \mathsf{poly}(\lambda).$$

The round complexity is $\Theta(d/\log s \cdot s^{1/c})$.

Finally, the following result shows the communication complexity of our MPC protocol when the succinct HSS scheme is instantiated by applying Theorem 7 on the constructions of Section 4.

Corollary 8. *Assume that one of the following holds:*

- *DCR over the Paillier group,*
- *DDH, hidden subgroup assumption, small exponent assumption, the uniformity assumption and $n^{1/3}$-ary EDDH over class groups,*
- *LWE with superpolynomial modulus-to-noise ratio.*

For any constant $c > 0$ and for every N-party functionality $\mathcal{F}$ represented by a layered boolean circuit C of size s, depth d, width n, with k inputs and m outputs, there exists a protocol that realises $\mathcal{F}$ against semi-honest adversaries corrupting up to $N-1$ parties in the plain model with total communication.

$$O(N \cdot (k + m + s/\log s)) + N^2 \cdot (d/\log s) \cdot s^{1/c} \cdot n^{2/3} \cdot \mathsf{poly}(\lambda).$$

The round complexity is $\Theta(d/\log s \cdot s^{1/c})$.

References

ADOS22. Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Heidelberg, August 2022.

Ajt96. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

BCG+19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

BCM23. Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear-communication secure multiparty computation does not require FHE. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 159–189. Springer, Heidelberg, April 2023.

BDG+22. Elette Boyle, Itai Dinur, Niv Gilboa, Yuval Ishai, Nathan Keller, and Ohad Klein. Locality-preserving hashing for shifts with connections to cryptography. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, pages 27–1. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2022.

BG10. Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2010.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016.

BGI+18. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018.

BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Heidelberg, May 2019.

BMRS21. Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 2021. Springer, Heidelberg.

CLM23. Valerio Cini, Russell W. F. Lai, and Giulio Malavolta. Lattice-based succinct arguments from vanishing polynomials - (extended abstract). In *CRYPTO 2023, Part II*, LNCS, pages 72–105. Springer, Heidelberg, August 2023.

CNs07. Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 573–590. Springer, Heidelberg, May 2007.

Cou19. Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 473–503. Springer, Heidelberg, May 2019.

DGI+19. Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019.

DIJL23. Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear mpc from sparse lpn. Springer-Verlag, 2023.

DIO21. Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

GHO20. Sanjam Garg, Mohammad Hajiabadi, and Rafail Ostrovsky. Efficient range-trapdoor functions and applications: Rate-1 OT and more. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 88–116. Springer, Heidelberg, November 2020.

GJM03. Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In Matt Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 120–135. Springer, Heidelberg, March 2003.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

IKM+13. Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 600–620. Springer, Heidelberg, March 2013.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

NN01. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *33rd ACM STOC*, pages 590–599. ACM Press, July 2001.

OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

QWW18. Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

Ros22. Razvan Rosie. Adaptively secure laconic function evaluation for NC^1. In Steven D. Galbraith, editor, *CT-RSA 2022*, volume 13161 of *LNCS*, pages 427–450. Springer, Heidelberg, March 2022.

RR22. Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2505–2517. ACM Press, November 2022.

RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Heidelberg.

Weg00. Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. Society for Industrial and Applied Mathematics, 2000.

Yao83. Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments (extended abstract). In *24th FOCS*, pages 420–428. IEEE Computer Society Press, November 1983.

YSWW21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021.