

Pairing-based Succinct Half-Chosen VOLE

Liqliang Liu, Tianren Liu, Luojuan Wei

Peking University

Abstract. Succinct Non-Interactive Half-Chosen VOLE allows a sender with a vector $\mathbf{x}$ of length n and a receiver with a scalar y to establish a pair of additive shares of $\mathbf{x} \cdot y$, by simultaneous exchange of succinct messages. Recent works construct NIHC-VOLE with $O(n^{2/3}\lambda)$ communication from various assumptions (Abram, Roy and Scholl, Eurocrypt 24) and poly-logarithmic communication from LWE (Abram, Malavolta and Roy, STOC 25).

We explore how bilinear pairings can improve group-based NIHC-VOLE schemes.

- In the public setup setting, our scheme has online communication of $O(\sqrt{n} \log n)$ group elements. The scheme relies on a new BDDH-like assumption.
- In the designated receiver setting, our scheme has online communication of only 3 group elements and reusable offline communication of $O(n)$ group elements. The scheme relies on the bilinear power DDH assumption.

1 Introduction

Vector Oblivious Linear Evaluation (VOLE) is a fundamental primitive in secure multi-party computation (MPC). A sender, who has two input vectors $\mathbf{x}, \mathbf{u} \in \mathbb{F}^n$, and a receiver, who has a scalar $y \in \mathbb{F}$, interact to enable the receiver to establish a vector $\mathbf{w}$ such that $\mathbf{w} = \mathbf{x} \cdot y + \mathbf{u}$. The term “obliviousness” basically means that $\mathbf{x}$ is hidden from the receiver and y is hidden from the sender. The scheme is *succinct* if the communication is sublinear in n , the length of the vectors.

A random version of VOLE, where $\mathbf{x}, \mathbf{u}, \mathbf{w}$ are all pseudorandom, has been widely studied in the context of pseudorandom correlation generators (PCG) [BCGI18, BCG⁺19, BCG⁺20]. However, when we want to make the sender’s input $\mathbf{x}$ “chosen” instead of “random”, a linear number of messages must be transmitted to derandomize, which destroys succinctness. This work studies succinct half-chosen VOLE: the sender, who chooses an input vector $\mathbf{x} \in \mathbb{F}^n$, and a receiver, who inputs a scalar y , interact to establish additive shares of $\mathbf{x} \cdot y$. *Half-chosen* distinguishes it from standard VOLE, in which the sender chooses two vectors. Half-chosen is necessary for succinct communication.

The beautiful work of [ARS24] showed that succinct half-chosen VOLE can even be realised through *non-interactive* communication, which means the sender and receiver simultaneously exchange their messages. They named it non-interactive half-chosen VOLE (NIHC-VOLE).

Most impressively, NIHC-VOLE has seen extensive applications. Using succinct VOLE as a core component, [ARS24] build a Multi-party Distributed Point Function (MPDPF) scheme. DPF, since first proposed in [GI14, BGI15], has been shown to be extremely useful in secure multi-party computation. It is widely used in private information retrieval [GI14], private set intersection [TSS⁺20, DIL⁺20, GRS22, GRS23, GGM24], pseudorandom correlation generators [BCGI18, BCG⁺19, BCG⁺20], secure computation of functions with mixed-mode operations [BGI19, BCG⁺21, BDSS25], and sublinear communication MPC [Cou19, CM21, ARS24]. Furthermore, succinct non-interactive VOLE has also been shown to imply trapdoor hash functions with rate-1 encoding key [CHP25], which can be used to build rate-1 batch OT, optimal preprocessing PIR, lossy trapdoor functions, rate-1/2 PSI and fuzzy PSI, evaluation over encrypted data for branching programs, 2PC with one-side statistical privacy and communication sublinear to both the circuit size and one party’s input [CHP25, DGI⁺19]. Succinct VOLE is also a building block for zero-knowledge proofs [WYKW21, YSWW21, DIO21, BMRS21, BBD⁺23].

	Online Communication	Assumption
[ARS24]	$O(n^{2/3}\lambda)$	DCR/ QR/ DDH-like assumption in class groups
[AMR25]	$\text{poly}(\lambda)$	LWE
Thm 5	$O((n \log n)^{1/2}\lambda)$	Split-Inverse BDDH

Table 1. Our main results in the NIHC-VOLE setting.

	Online Communication	Reusable Offline Communication	Assumption
[ARS24]	$O(n^{1/2}\lambda)$	$O(n\lambda)$	n -Power DDH / n -Power RingLWE
[CHP25]	$O(\lambda)$	$O(n^2\lambda)$	[ARS24] + NC1 PRF
[GHO20]	$O(n + \lambda)$	$O(n\lambda)$	$2n$ -Power DDH
Thm 6	$O(\lambda)$	$O(n^{\log_2 3}\lambda)$	Bilinear 1-Power DDH
Thm 7	$O(\lambda)$	$O(n\lambda)$	Bilinear $(n + 1)$ -Power DDH

Table 2. Our main results in the NIHC-VOLE setting, in which the designated receiver has a trapdoor.

The best group-based NIHC-VOLE [ARS24] has sublinear communication $O(n^{2/3}\lambda)$. Recently, Abram, Malavolta, and Roy [AMR25] show how to construct NIHC-VOLE from LWE with only $\text{poly}(\lambda)$ communication. A natural open problem is to achieve similar communication from group assumptions.

We partially solved this problem by incorporating bilinear maps into this field. In a nutshell, we explore how to compress the receiver’s message through bilinear maps.

1.1 Our Contributions

Non-Interactive Half-Chosen VOLE (NIHC-VOLE). As our first contribution, we improve the NIHC-VOLE construction of [ARS24]. In our construction, the CRS size is $O(\sqrt{n/\log n\lambda})$, and the message size of both sender and receiver is $O(\sqrt{n\log n\lambda})$. The privacy of this scheme relies on a new assumption in bilinear groups, which we call Split-Inverse Bilinear DDH.

NIHC-VOLE with Preprocessing (pNIHC-VOLE). We propose a new variant called NIHC-VOLE with preprocessing, in which the receiver generates a relatively long *public reusable string* in the preprocessing phase. The receiver may store a secret trapdoor during preprocessing. This relaxation enables very succinct online communications.

In this model, we propose two constructions assuming the bilinear power DDH assumption:

1. Assuming bilinear 1-power DDH, the public reusable string is of size $O(n^{\log_2 3} \lambda)$ and the online communication is only $O(\lambda)$.
2. Assuming bilinear $(n + 1)$ -power DDH, the public reusable string is of size $O(n\lambda)$ and the online communication is only $O(\lambda)$.

Thus, after a one-time reusable preprocessing phase, the communication of each subsequent NIHC-VOLE execution is independent of the vector length n . Moreover, we explore trade-offs between the strength of the assumptions and the efficiency of these two schemes, which then allows us to discover a range of constructions between them.

It is worth mentioning that [CHP25] inherently constructs a pNIHC-VOLE scheme, which we elaborate in the technical overview.

1.2 Applications of Our Scheme

Equipped with our improved NIHC-VOLE schemes, we obtain the following applications. They generally follow from plugging our NIHC-VOLE into frameworks established in prior works [CHP25, ARS24]. Roughly speaking, plugging in our plain NIHC-VOLE is more efficient than existing group-based construction, but usually offers no asymptotic communication advantage over lattice-based construction [AMR25]. On the other hand, our NIHC-VOLE with preprocessing schemes yields remarkably short online communication, superior even to [AMR25].

Succinct HSS with Public Setup. Informally, a succinct HSS with public setup allows two parties, Alice (with $\mathbf{x}$) and Bob (with $\mathbf{y}$), after the public setup phase, to additively share $P(\mathbf{x}, \mathbf{y})$ by sending simultaneous messages to each other. It is noteworthy that the function P can be chosen on the fly. Most impressively, as pointed out by [CHP25], succinct HSS with public setup can be

used to build efficient trapdoor hash functions (TDH), which have been shown to have extensive applications.

Recently, [CHP25] constructed a succinct HSS with public setup that costs $O(n^{2/3} \text{poly}(\lambda) + m)$ bits of (online) communication, where $m = |\mathbf{x}|$ and $n = |\mathbf{y}|$ are the input lengths. Their scheme supports any P that is a bilinear RMS program, i.e., P can be represented as $P(\mathbf{x}, \mathbf{y}) = \left(\sum_{j \in [\ell]} F_{i,j}(\mathbf{x}) \cdot G_{i,j}(\mathbf{y}) \right)_{i \in [\ell_{\text{out}}]}$ where $F_{i,j}$ and $G_{i,j}$ are all RMS programs¹. Their construction relies on the DCR or DDH assumption.

Combining known HSS techniques with our improved NIHC-VOLE scheme, we construct an improved succinct HSS with public setup from the Split-Inverse BDDH assumption. Our improvements lie in two facets. (1) the communication complexity is $O(\sqrt{n \log n} \text{poly}(\lambda) + m)$. (2) our scheme extends the class of P to bilinear half-RMS programs, i.e., $P(\mathbf{x}, \mathbf{y}) = \left(\sum_{j \in [\ell]} F_{i,j}(\mathbf{x}) \cdot G_{i,j}(\mathbf{y}) \right)_{i \in [\ell_{\text{out}}]}$, where $F_{i,j}$ is a RMS program and $G_{i,j}$ is any circuit.²

Our improved HSS scheme directly gives a TDH scheme for bilinear half-RMS programs that has encoding key size $\ell + \sqrt{\ell \log \ell} \cdot \text{poly}(\lambda)$ and hash digest size $\sqrt{\ell \log \ell} \cdot O(\lambda)$, where ℓ is the input length. This improves [CHP25], whose key size and hash digest size are $\ell + \ell^{2/3} \cdot \text{poly}(\lambda)$ and $\ell^{2/3} O(\lambda)$, respectively. Numerous applications follow from TDH, including batch-OT, PIR, PSI, and sublinear 2PC. One can check [DGI⁺19, CHP25] for those applications. Our succinct HSS and TDH constructions improve these applications in a plug-and-play manner.

We remind that one can achieve $(\text{poly}(\log n, \lambda) + m)$ communication if the underlying NIHC-VOLE is instantiated by the lattice-based construction of [AMR25]. We also note that a recent work [MORS26] inherently constructed a succinct HSS scheme for bilinear half-RMS programs, as a by-product of their bundled homomorphic MAC scheme. They achieved roughly $O(n^{2/3} \text{poly}(\lambda) + m)$ communication under DCR, and enjoyed an additional computationally succinct property of Bob. In contrast, our work centers on communication optimization, and our technique differs substantially from theirs.

Succinct HSS with Reusable Public Key. Succinct HSS with a reusable public key is a variant of succinct HSS with public setup. The major difference is that, instead of outputting simply the CRS as in the public setup mode, the non-public setup algorithm outputs a (possibly large) reusable public key, an evaluation key, and a private key. Bob can only see the reusable public key and evaluation key, and Alice gets the private key.

Combining our pNIHC-VOLE and known HSS techniques, we construct several succinct HSS with reusable public key schemes, also for bilinear half-RMS

¹ RMS programs are a special class of circuits, where multiplications are only allowed between inputs and memory values, and additions are not restricted.

² To realize succinct HSS for bilinear half-RMS programs, we additionally employ the algebraic homomorphic MAC (aHMAC) scheme proposed by [ILL25a]. aHMAC can be realized based on various assumptions, including the circular power DDH assumption. See the technical overview and appendix for details.

programs ³: (1) assuming bilinear 1-power DDH, the reusable public key size is $O(n^2\lambda) + \text{poly}(\lambda)$ (or $O(n^{\log_2 3}\lambda) + \text{poly}(\lambda)$, improved) and the communication is $m + \text{poly}(\lambda)$; (2) assuming bilinear $(n + 1)$ -power DDH, the reusable public key size is $O(n\lambda) + \text{poly}(\lambda)$ and the communication is $m + \text{poly}(\lambda)$. In short, the communication complexity is independent of Bob’s input size, and is rate-1 to Alice’s input size. We also remark that the public key is reusable not only for multiple inputs of the HSS, but also for multiple generations of the evaluation key and private key.

Although this variant of succinct HSS has not been known to imply TDH, we still find it interesting as an independent model, and it’s very likely to enable future applications in PIR, OT, or 2PC due to its super-succinct online communication.

Multi-party Distributed Point Function. A point function $F_{\alpha,\beta}$ is a function that evaluates to β at the point α and vanishes to zero elsewhere. A multi-party distributed point function (MPDPF) scheme allows a trusted dealer with input α, β to generate a succinct key for each party. These keys serve as succinct secret sharings of $F_{\alpha,\beta}$: the parties can locally derive additive shares of $F_{\alpha,\beta}(x)$ for each point x in the domain. MPDPF is very useful in secure multi-party computation, especially in building PCGs [BCGI18, BCG⁺19, BCG⁺20] and sublinear MPC [Cou19, CM21].

To the best of our knowledge, for N -party DPF against $(N - 1)$ -corruptions with domain $\mathcal{D}$ and arbitrary N , the state-of-the-arts are (1) the construction of [GWW25] with key size $O(|\mathcal{D}|^{1/2+\epsilon} N^3 \text{poly}(\lambda))$ from OWF; (2) the construction of [ARS24] with key size $O(|\mathcal{D}|^{2/3} N \text{poly}(\lambda))$ from DCR, DDH, LWE or class groups; (3) the construction of [ARS24] with key size $O(|\mathcal{D}|^{1/2} N \text{poly}(\lambda))$ from power-DDH or power-RingLWE. A recent work of [CKY25] also constructed an MPDPF with key size polylogarithmic in the domain size from LPN and MQ, but only for a constant number of parties.

By applying the general framework of constructing MPDPF from NIHC-VOLE [ARS24], we shrink the key. From our NIHC-VOLE scheme based on Split-Inverse BDDH, we construct an N -party DPF against static, semi-honest $(N - 1)$ corruptions, with key size $O(\sqrt{|\mathcal{D}|} \log |\mathcal{D}| N \text{poly}(\lambda))$. We remind that if one adopts the NIHC-VOLE scheme of [AMR25] from lattices, the key size will be $O(N \cdot \text{poly}(\log |\mathcal{D}|, \lambda))$.

We also consider a new variant of MPDPF, named MPDPF in the CRS model. In this model, the dealer first generates a large (structured, public) CRS before it determines α, β . Later, it generates DPF keys, which are very succinct, from α, β and the CRS. Furthermore, each party’s key has two components: a part which is reusable in different MPDPF instantiations, and a super-succinct part which will be sent each time.

From our pNIHC-VOLE scheme, we build an N -party DPF against static, semi-honest $(N - 1)$ -corruptions in this CRS model, assuming bilinear power-DDH, with the CRS size $O(N(|\mathcal{D}|\lambda + \text{poly}(\lambda)))$, the reusable key size $O(N \text{poly}(\lambda))$,

³ We also assume the existence of aHMAC.

and the fresh key size $O(N(\log |\mathcal{D}| + \lambda))$. This online key size is extremely succinct even in comparison with the lattice-based construction.

On the Correctness Error and Concrete Efficiency. One caveat that we have not mentioned yet is: similar to the constructions based on prime-order groups in our precedents, we invoke the distributed discrete logarithm (DDLog) algorithm for prime-order groups [BGI16] to turn multiplicative shares into additive shares. This triggers an inverse-poly correctness error and works only when the magnitude of $\mathbf{x} \cdot \mathbf{y}$ is sufficiently small. In other words, we handle small-field VOLE.

In terms of concrete efficiency, the security-parameter dependence in our communication complexity is rather straightforward, in contrast to lattice-based precedents, in which a substantial polynomial factor is concealed behind the asymptotic notation. The computations of our pairing-based constructions are more efficient than the lattice-based counterparts when the cost of DDLog is not taken into account. Unfortunately, decreasing the cost of the DDLog in cyclic groups will lead to a higher correctness error rate [BGI16]. It's left open to do efficient error correction in the non-interactive setting. However, it's generally simple by appending an extra unilateral message⁴. Thus, the MPDPF we construct in this paper can be efficient. It does not require non-interactivity, as one can see shortly.

1.3 Technical Overview

Recap the Construction of [ARS24]. Their technique builds on the TDH construction of [DGI⁺19]. Suppose the sender holds a vector $\mathbf{x}$ of length n , and the receiver holds a scalar y . For a group $\mathbb{G}$ of order q generated by g , we use $[x]_{\mathbb{G}}$ to denote g^x . Set the CRS as $n + 1$ random group elements:

$$\text{common reference string} = [a_0]_{\mathbb{G}}, [a_1]_{\mathbb{G}}, \dots, [a_n]_{\mathbb{G}}.$$

The sender's message is simply a hash of $\mathbf{x}$, and one-time padded by a fresh random value r :

$$\begin{aligned} \text{sender's hash value} &= ([a_0]_{\mathbb{G}})^r ([a_1]_{\mathbb{G}})^{x_1} \dots ([a_n]_{\mathbb{G}})^{x_n} \\ &= [a_0 r + a_1 x_1 + \dots + a_n x_n]_{\mathbb{G}}. \end{aligned}$$

The receiver's message is a $(n + 1) \times n$ matrix as the encryption of $y \cdot \mathbf{I}_{n \times n}$:

$$\mathbf{M} := \begin{pmatrix} ([a_0]_{\mathbb{G}})^{b_1} & ([a_0]_{\mathbb{G}})^{b_2} & \dots & ([a_0]_{\mathbb{G}})^{b_n} \\ ([a_1]_{\mathbb{G}})^{b_1} \cdot [y]_{\mathbb{G}} & ([a_1]_{\mathbb{G}})^{b_2} & \dots & ([a_1]_{\mathbb{G}})^{b_n} \\ ([a_2]_{\mathbb{G}})^{b_1} & ([a_2]_{\mathbb{G}})^{b_2} \cdot [y]_{\mathbb{G}} & \dots & ([a_2]_{\mathbb{G}})^{b_n} \\ \vdots & \vdots & \ddots & \vdots \\ ([a_n]_{\mathbb{G}})^{b_1} & ([a_n]_{\mathbb{G}})^{b_2} & \dots & ([a_n]_{\mathbb{G}})^{b_n} \cdot [y]_{\mathbb{G}} \end{pmatrix}.$$

⁴ There are previous works discussing error correction for interactive protocols [BGI17, CHP25, ILL25b]. A more accessible exposition is also provided in the technical overview.

Now the sender can take the i -th column and compute

$$\begin{aligned} v_i^S &:= ([a_0 b_i]_{\mathbb{G}})^r \cdot ([a_1 b_i]_{\mathbb{G}})^{x_1} \cdots ([a_i b_i + y]_{\mathbb{G}})^{x_i} \cdots ([a_n b_i]_{\mathbb{G}})^{x_n} \\ &= [(a_0 r + a_1 x_1 + \cdots + a_n x_n) b_i + y x_i]_{\mathbb{G}} \text{ for } i \in \{1, \dots, n\}. \end{aligned}$$

The receiver can take the hash value and private randomness b_i and compute

$$v_i^R := [(a_0 r + a_1 x_1 + \cdots + a_n x_n) b_i]_{\mathbb{G}} \text{ for } i \in \{1, \dots, n\}.$$

Now that $(v_i^S, v_i^R)_{i=1}^n$ form divisional shares of $g^{x \cdot y}$, they invoke the standard distributed discrete logarithm (DDLog) algorithm [BGI16] locally to obtain subtractive shares of $x \cdot y$. The correctness of this algorithm holds (with an inverse-poly correctness error) when the magnitude of x and y are sufficiently small. Thus the above construction (and also our improvements) only deals with small-field VOLE.

Note that in the construction above, the sender's message has $O(\lambda)$ bit length while the receiver's message has $O(n^2 \lambda)$ bit length, which is unbalanced. The key observation is that by reusing simultaneous messages, the sender can divide x into $n^{2/3}$ segments of length $n^{1/3}$: $x[1 : n^{1/3}], x[n^{1/3} + 1 : 2n^{1/3}], \dots, x[n - n^{1/3} + 1 : n]$. The receiver hashes each of them, for a total size of $O(n^{2/3} \lambda)$. The sender now only needs to send a $(n^{1/3} + 1) \times n^{1/3}$ matrix as the encryption of $y \cdot I_{n^{1/3} \times n^{1/3}}$. In this case, the sender and receiver's message sizes are balanced, both of $O(n^{2/3} \lambda)$.

Improved NIHC-VOLE from pairings. The balancing trick suggests that compressing the receiver's ciphertext matrix $\mathbf{M}$ will improve communication complexity. Moreover, the receiver needs only to encrypt a single y , so the matrix $\mathbf{M}$ appears redundant. We manage to shrink the description size of $\mathbf{M}$ from $O(n^2 \lambda)$ to $O(n \log n \lambda)$ by cleverly using pairings.

Consider a *bilinear group* $\mathbb{G}_T$ with *bilinear map* $e : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ such that $e(g^a, h^b) = g_T^{ab}$. Observe that by using g_T as the generator in the above construction, all entries of $\mathbf{M}$ without y can be compressed into only $O(n)$ elements $[a_i]_{\mathbb{G}}, [b_j]_{\mathbb{H}}$. Since only n entries contain y , the ciphertext can be compressed into $([a_i]_{\mathbb{G}}, [b_j]_{\mathbb{H}}, [a_k b_k + y]_{\mathbb{G}_T})$ for $i \in \{0, 1, \dots, n\}, j, k \in \{1, 2, \dots, n\}$.

However, this straightforward attempt fails since $[a_1 b_1 + y]_{\mathbb{G}_T}, [a_1]_{\mathbb{G}}, [b_1]_{\mathbb{H}}$ leaks $[y]_{\mathbb{G}_T}$, violating the security requirement. To protect y given $[a_k b_k + y]_{\mathbb{G}_T}$, we introduce $2\ell = 2(\lfloor \log_2 n \rfloor + 1)$ random modifiers $s_{1,0}, s_{1,1}, \dots, s_{\ell,0}, s_{\ell,1}$. Instead of sending $[a_i]_{\mathbb{G}}, [b_j]_{\mathbb{H}}$ directly, we send modified elements

$$\begin{aligned} &[a_i \cdot s_{1, \text{Bin}_1(i)}]_{\mathbb{G}}, \dots, [a_i \cdot s_{\ell, \text{Bin}_\ell(i)}]_{\mathbb{G}}, \quad \text{for } i \in \{0, 1, \dots, n\}, j \in \{1, 2, \dots, n\}, \\ &[b_j / s_{1, 1 - \text{Bin}_1(j)}]_{\mathbb{H}}, \dots, [b_j / s_{\ell, 1 - \text{Bin}_\ell(j)}]_{\mathbb{H}} \end{aligned}$$

where $\text{Bin}_k(i)$ denotes the k -th significant bit in binary representation of i . Intuitively, for $i \neq j$, there must be some k such that $\text{Bin}_k(i) = 1 - \text{Bin}_k(j)$, one can recover from the modified elements

$$[a_i b_j]_{\mathbb{G}_T} = e([a_i \cdot s_{k, \text{Bin}_k(i)}]_{\mathbb{G}}, [b_j / s_{k, 1 - \text{Bin}_k(j)}]_{\mathbb{H}}).$$

For $i = j$, no such k exists, hence $[a_i b_i]_{\mathbb{G}_T}$ remains hidden from the sender (this hidden property relies on our new assumption named Split-Inverse Bilinear DDH). The total ciphertext size is $O(n \log n \lambda)$, and the total communication is $O(\sqrt{n \log n \lambda})$ after balancing.

NIHC-VOLE with Preprocessing. Given that the receiver's message y is short, even compressing the ciphertext size to $n \log n$ group elements seems far from optimal. Ideally, a constant number of group elements should suffice to hide y . We show that this is possible, at the cost of a relatively heavy but reusable preprocessing phase.

Observe that the number of elements that depend on y (and thus cannot be prepared during the preprocessing phase) in $\mathbf{M}$ is n . To cover them with a constant number of elements, we restrict $a_i b_i = t$ for $i \in \{1, \dots, n\}$ and some global t . Under this constraint, for all $i \in \{1, \dots, n\}$, $[a_i b_i + y]_{\mathbb{G}_T}$ are equal and cost only one group element of communication.

Replace b_i with ta_i^{-1} and write $\mathbf{M}$ as

$$\begin{pmatrix} [ta_0 a_1^{-1}]_{\mathbb{G}_T} & [ta_0 a_2^{-1}]_{\mathbb{G}_T} & \cdots & [ta_0 a_n^{-1}]_{\mathbb{G}_T} \\ [t + y]_{\mathbb{G}_T} & [ta_1 a_2^{-1}]_{\mathbb{G}_T} & \cdots & [ta_1 a_n^{-1}]_{\mathbb{G}_T} \\ [ta_2 a_1^{-1}]_{\mathbb{G}_T} & [t + y]_{\mathbb{G}_T} & \cdots & \vdots \\ \vdots & \vdots & \ddots & [ta_{n-1} a_n^{-1}]_{\mathbb{G}_T} \\ [ta_n a_1^{-1}]_{\mathbb{G}_T} & \cdots & [ta_n a_{n-1}^{-1}]_{\mathbb{G}_T} & [t + y]_{\mathbb{G}_T} \end{pmatrix}.$$

The off-diagonal elements in $\mathbf{M}$ are independent of y , let the receiver prepare a matrix $\mathbf{P}$ as public reusable string (prs)

$$\mathbf{P} := \begin{pmatrix} [t_0 a_0 a_1^{-1}]_{\mathbb{G}} & [t_0 a_0 a_2^{-1}]_{\mathbb{G}} & \cdots & [t_0 a_0 a_n^{-1}]_{\mathbb{G}} \\ \times & [t_0 a_1 a_2^{-1}]_{\mathbb{G}} & \cdots & [t_0 a_1 a_n^{-1}]_{\mathbb{G}} \\ [t_0 a_2 a_1^{-1}]_{\mathbb{G}} & \times & \cdots & \vdots \\ \vdots & \vdots & \ddots & [t_0 a_{n-1} a_n^{-1}]_{\mathbb{G}} \\ [t_0 a_n a_1^{-1}]_{\mathbb{G}} & \cdots & [t_0 a_n a_{n-1}^{-1}]_{\mathbb{G}} & \times \end{pmatrix}.$$

To reconstruct $\mathbf{M}$, the sender with prs only needs $[t/t_0]_{\mathbb{H}}$ and $[t + y]_{\mathbb{G}_T}$ in the online phase—just compute $e(\mathbf{P}, [t/t_0]_{\mathbb{H}})$, and fill in the blanks with $[t + y]_{\mathbb{G}_T}$. The process after constructing $\mathbf{M}$ remains identical to the original scheme. The prs size is $O(n^2 \lambda)$ and the online communication consists of only 3 group elements.

Is $\mathbf{P}$ still compressible? We answer it positively by sampling a_i from delicate distributions and creating redundancy within $\mathbf{P}$. Note that if for some $(i, j) \neq (k, w)$, we have $a_i a_j^{-1} = a_k a_w^{-1}$, then the prs size can be reduced further since some duplicated entries in $\mathbf{P}$ can be sent together. By choosing a_i with structure such that $a_i = \prod_{j=1}^{\ell} b_j^{\text{Bin}_j(i)}$ or $a_i = \alpha^i$, we present two optimized schemes with the prs size $O(n^{\log_2 3} \lambda)$ and $O(n \lambda)$, respectively. These two schemes rely on the bilinear power DDH assumptions with different parameters, and the online communication remains 3 group elements.

An Implicit Construction of pNIHC-VOLE by [CHP25]. We thank the CRYPTO 26 reviewers for pointing it out. [CHP25] develops a trick that converts an NIHC-VOLE with highly unbalanced communication to a pNIHC-VOLE with succinct online communication.

We have shown at the beginning of this section that [ARS24] gives an NIHC-VOLE with the sender’s message of 1 group element and the receiver’s message of n^2 group elements. Imagine that the two parties run such an NIHC-VOLE for $\mathbf{x} \cdot s$, where s is a secret key sampled by the receiver. For the preprocessing, only the receiver’s message is sent, which is input-independent. The receiver also sends an encryption of a PRF key sk for a public NC1 PRF, with the secret key s . In the online phase, the sender sends the 1-group-element hash for the NIHC-VOLE between $\mathbf{x}$ and s . This enables them to derive additive shares of $\mathbf{x} \cdot s$. Meanwhile, the receiver also sends $y' := \text{PRF}(\text{sk}, id) + y$. Finally, they run extended HSS⁵ evaluations on $\text{Enc}(s, \text{sk})$ and shares of $\mathbf{x} \cdot s$, for the program $y' - \text{PRF}(\cdot, id)$. They locally establish shares of $\mathbf{x} \cdot y$. This approach requires non-black-box HSS evaluation of a PRF.

Applications: Succinct HSS. In general, our constructions of succinct HSS (with public setup or with reusable public key) for bilinear half-RMS programs follow the template of [ARS24, CHP25]. Within our construction, we borrow techniques from recent works on HSS. We briefly overview them below:

- Algebraic Homomorphic MAC (aHMAC) [ILL25a]: Suppose Alice has a secret key s and Bob has an input vector $\mathbf{y}$. Additionally, they have subtractive shares of $s \cdot \mathbf{y}$. Then by letting Alice send a reusable evaluation key to Bob, Alice and Bob can locally derive shares of $s \cdot C(\mathbf{y})$ for any circuit C . In [ILL25a], they proposed constructions from the circular power DDH assumption in Damgård-Jurik groups, class groups, or prime order groups, or the KDM-DCR assumption. The evaluation key is of size $\text{poly}(\lambda)$.
- Extended HSS Evaluation [CMR23]: In standard HSS evaluation for RMS programs, the inputs are formed as encryptions. An encryption can be transferred into subtractive shares, and multiplication is feasible between shares and encryption. Overall, given encryptions of $\mathbf{x}$, the participants can locally obtain shares of $F(\mathbf{x})$, for RMS program F . The extended HSS states that, if they additionally have a pair of shares of $s \cdot \mathbf{y}$, where s is the secret key of HSS, they could locally obtain shares of $\mathbf{y} \cdot F(\mathbf{x})$.

With these techniques in place, we now introduce our construction of succinct HSS with public setup. Alice holds input $\mathbf{x}$, and Bob holds input $\mathbf{y}$. During the setup phase, the protocol generates a CRS as in our VOLE scheme. Then Alice samples a secret key s and runs an NIHC-VOLE with Bob, from which they obtain subtractive shares of $s \cdot \mathbf{y}$ ⁶. Meanwhile, Alice sends the evaluation key

⁵ This gadget will be explained shortly.

⁶ Our NIHC-VOLE only works for small fields. Thus, in the formal construction, VOLE is done between each bit of the binary decomposition of s and $\mathbf{y}$. aHMAC also handles this vector version of the secret key.

of the aHMAC to Bob, which enables them to locally derive shares of $s \cdot G(\mathbf{y})$, for any circuit G . Simultaneously, Alice sends encryptions of $\mathbf{x}$, which consists of two parts: an HSS encryption of a PRF key k , and a one-time pad $\mathbf{x}'$ of $\mathbf{x}$ by $\text{PRF}(k, 1), \dots, \text{PRF}(k, |\mathbf{x}|)$. Finally, they locally run the extended HSS evaluation on shares of $s \cdot G(\mathbf{y})$, encryptions of PRF key k , for the RMS program $F'(k) := F(\mathbf{x}' - (\text{PRF}(k, i))_{i=1}^{|\mathbf{x}|})$. In the end, they get shares of $F(\mathbf{x}) \cdot G(\mathbf{y})$.

The above protocol is one-round, simultaneous-message. With our improved NIHC-VOLE from Split-Inverse BDDH, the communication complexity is $O(\sqrt{|\mathbf{y}|} \log |\mathbf{y}| \text{poly}(\lambda) + |\mathbf{x}|)$.

Our construction of succinct HSS with reusable public key is similar. The key difference is using pNIHC-VOLE instead of NIHC-VOLE. The public reusable string in our VOLE construction now serves as the reusable public key. The benefit is a very succinct (non-interactive) online phase: the communication is $O(|\mathbf{x}| + \text{poly}(\lambda))$ —independent of $\mathbf{y}$ and rate-1 to $\mathbf{x}$.

Applications: Multi-party DPF. We essentially adopt the [ARS24] transformation from a succinct HSS for special RMS programs to an MPDPF. A special RMS program is defined as $P(\mathbf{x}, \mathbf{y}) = (\langle F_i(\mathbf{x}), \mathbf{y} \rangle)_{i=1}^{\ell_{\text{out}}}$ where F_i is an RMS program and $\langle \cdot, \cdot \rangle$ denotes inner product. In other words, $P(\mathbf{x}, \mathbf{y})$ is linear in $\mathbf{y}$. One can easily tell that special RMS programs are degenerations of bilinear half-RMS programs. Our constructions of succinct HSS work naturally for special RMS programs by removing the aHMAC procedure.

At a high level, the dealer first runs a multi-round, interactive MPDPF protocol in its head, and takes each party's view as the DPF key. As long as the views in the protocol are succinct, the MPDPF keys will be succinct. The multi-round protocol runs recursively. Suppose the N parties relatively holds $\alpha_1, \dots, \alpha_N$ with $\sum \alpha_i = \alpha$ ⁷. We use notation $\mathbf{p}(\alpha)$ to denote the vector with non-zero only at position α . Now suppose that the first k parties have already got additive shares of $\mathbf{p}(\sum_{i=1}^k \alpha_i)$, say relatively $\langle \mathbf{p}(\sum_{i=1}^k \alpha_i) \rangle_1, \dots, \langle \mathbf{p}(\sum_{i=1}^k \alpha_i) \rangle_k$. Then party $(k+1)$, taking α_{k+1} as input, runs a succinct HSS scheme with each of parties $1, \dots, k$. This relies on the fact that “shifting vector $\langle \mathbf{p}(\sum_{i=1}^k \alpha_i) \rangle_j$ by α_{k+1} ” is a special RMS program (linear with $\langle \mathbf{p}(\sum_{i=1}^k \alpha_i) \rangle_j$). Finally, set the view of each party in this multi-round protocol as its MPDPF key.

Equipped with our succinct HSS with public setup for special RMS programs, each party's key size is $O(N \sqrt{|\mathcal{D}|} \log |\mathcal{D}| \text{poly}(\lambda))$.

Built upon our succinct HSS with reusable public key, we additionally get an MPDPF in the CRS model. Within each two-party interaction in the multi-round protocol that runs in the dealer's head: (1) the reusable public key can be published and reused, and is of size $O(|\mathcal{D}| \lambda)$; (2) the evaluation key is input-independent, and thus can also be published and reused, and is of size $\text{poly}(\lambda)$; (3) the secret key is private, but input-independent (thus reusable), and is of size $\text{poly}(\lambda)$; (4) the head of party $(k+1)$'s ciphertext that encrypts the PRF key is

⁷ For simplicity, we ignore the payload β and only consider the non-zero position α for now. Because sharing the position is much harder than sharing the payload

reusable, and is of size $\text{poly}(\lambda)$; (5) the one-time pads of α_{k+1} is non-reusable, and is of size $\log |\mathcal{D}|$; (6) the hash value from party $1, \dots, k$ is non-reusable, and is of size $O(\lambda)$. To summarize, the structured CRS size is $O(N(|\mathcal{D}|\lambda + \text{poly}(\lambda)))$; the reusable part of each key is of size $O(N \text{poly}(\lambda))$; the fresh part of each key is of size $O(N(\log |\mathcal{D}| + \lambda))$.

Dealing with Errors. The DDH-based DDLog incurs a $1/\text{poly}(\lambda)$ correctness error. We borrow ideas from [BG17] and [CHP25] to amplify the error to $\text{negl}(\lambda)$, at the expense of an extra round of unilateral messages.

We first describe the DDLog process. The two parties agree on a PRF key k , which defines a set of “distinguished points” in the cyclic group $\mathbb{G}$, defined as $S_k := \{p : p \in \mathbb{G}, \text{PRF}(k, p) = 0\}$. Suppose Alice and Bob hold shares h and $h \cdot g^{xy}$, where $x, y \in \{0, 1\}$ are Alice’s and Bob’s inputs, respectively. DDLog outputs the gap in the exponent between their share and a nearest distinguished point. Correctness fails if Alice reaches a distinguished point different from the one reached by Bob. Thus, the correctness error is roughly the inverse of the average gap between distinguished points, which is $1/\text{poly}(\lambda)$.

The key observation is that, since $xy \in \{0, 1\}$, correctness holds if h is not a distinguished point. The first idea is to let Alice and Bob agree on multiple sets (instead of one), i.e., multiple PRF keys $k_1, \dots, k_t$. For $t = o(\lambda)$, it holds with overwhelming probability that $\exists i \in \{1, \dots, t\}$ such that $h \notin S_{k_i}$. Then Alice simply sends i to Bob, so that they agree on a common set for DDLog.

The problem with this method is that revealing Alice’s “comfortable” set leaks x —if $y = 1$ and $h \cdot g^{xy}$ is one- g ahead of a distinguished point in S_{k_i} , Bob learns $x = 0$. However, this leakage happens with probability $1/\text{poly}(\lambda)$. Alice can secret-share her input to $x_1, \dots, x_\ell$, and repeat the process above for each $x_j y$. For $\ell = O(\lambda)$, the probability that x is leaked will be negligible.

The total communication overhead is $o(\lambda)$, and it requires an extra message from Alice to Bob (or vice versa), which undermines the non-interactivity. However, some applications of half-chosen VOLE, such as multi-party DPF, do not require the non-interactive property of VOLE.

2 Preliminaries and Definitions

2.1 Notations

We use $[n]$ to denote the set $\{0, 1, \dots, n-1\}$, and use $[n]^*$ to denote the set $\{1, \dots, n-1\}$. For a string or vector $\mathbf{v}$ with length ℓ , we use $\mathbf{v}[i]$ to denote its i -th entry, i.e. $\mathbf{v} = (\mathbf{v}[1], \dots, \mathbf{v}[\ell])$. For a binary string $\mathbf{s}$, we define $\bar{\mathbf{s}}$ such that $\bar{\mathbf{s}} \oplus \mathbf{s} = 11 \dots 1$. For two vectors $\mathbf{a}$ and $\mathbf{b}$, we use $\langle \mathbf{a}, \mathbf{b} \rangle$ to denote their inner product. For two matrices $\mathbf{A}$ and $\mathbf{B}$, we use $\mathbf{A} \otimes \mathbf{B}$ to denote their standard tensor product.

We use $\text{Bin}(\cdot)$ to denote the bit decomposition function, and $\text{BC}(\cdot)$ to denote the bit composition function (the inverse function of Bin). That is, $\text{Bin}(x) \in \{0, 1\}^{\lceil \log_2 x \rceil + 1}$, and $\text{BC}(\text{Bin}(x)) = x = \sum_{i=1}^{\lceil \log_2 x \rceil + 1} \text{Bin}(x)[i] \cdot 2^{i-1}$. We use $\text{Bin}_j(x)$ to denote the j -th significant bit in binary representation of x .

2.2 Computational Assumptions

A *group* is described by a tuple $(q, g, \mathbb{G})$. $\mathbb{G}$ has prime order q , and generator g . We denote $[x]_{\mathbb{G}} := g^x$.

Assumption 1 (Decisional Diffie-Hellman) *In a given group $(q, g, \mathbb{G})$,*

$$(q, g, \mathbb{G}, [x]_{\mathbb{G}}, [y]_{\mathbb{G}}, [xy]_{\mathbb{G}}) \stackrel{c}{\approx} (q, g, \mathbb{G}, [x]_{\mathbb{G}}, [y]_{\mathbb{G}}, [r]_{\mathbb{G}})$$

where $x, y, r \xleftarrow{\$} \mathbb{Z}_q^*$.

Assumption 2 (k -Power DDH) *In a given group $(q, g, \mathbb{G})$,*

$$(q, g, \mathbb{G}, [\alpha^{-k}x]_{\mathbb{G}}, \dots, [\alpha^{-1}x]_{\mathbb{G}}, [x]_{\mathbb{G}}, [\alpha x]_{\mathbb{G}}, \dots, [\alpha^k x]_{\mathbb{G}}) \\ \stackrel{c}{\approx} (q, g, \mathbb{G}, [r_{-k}]_{\mathbb{G}}, \dots, [r_{-1}]_{\mathbb{G}}, [r_0]_{\mathbb{G}}, [r_1]_{\mathbb{G}}, \dots, [r_k]_{\mathbb{G}})$$

where $x, \alpha, r_{-k}, \dots, r_0, \dots, r_k \xleftarrow{\$} \mathbb{Z}_q^*$.

We remind that some traditional formulations of power DDH state that given $([x]_{\mathbb{G}}, [\alpha x]_{\mathbb{G}}, \dots, [\alpha^k x]_{\mathbb{G}})$, it's hard to distinguish $[\alpha^{k+1}x]_{\mathbb{G}}$ or $[\alpha^{-1}x]_{\mathbb{G}}$ from random [GJM03, Cns07, AMN⁺18]; or that given $([\alpha^{-k}x]_{\mathbb{G}}, \dots, [\alpha^{-1}x]_{\mathbb{G}}, [\alpha x]_{\mathbb{G}}, \dots, [\alpha^k x]_{\mathbb{G}})$, it's hard to distinguish $[x]_{\mathbb{G}}$ from random [CHHK25]. As shown in [CHHK25], the variant formulated in Assumption 2 is equivalent to other variants of power DDH assumptions up to a factor-2 loss in the parameter k .

Bilinear groups are described by a tuple $(q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e)$. $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ are groups of prime order q , and g, h, g_T are respectively their generators. We use the notation $[x]_{\mathbb{G}} := g^x, [y]_{\mathbb{H}} := h^y, [z]_{\mathbb{G}_T} := g_T^z$. $e : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ is an efficiently computable function such that $e([x]_{\mathbb{G}}, [y]_{\mathbb{H}}) = [xy]_{\mathbb{G}_T}$. For two matrices $[\mathbf{A}]_{\mathbb{G}}$ in $\mathbb{G}$ and $[\mathbf{B}]_{\mathbb{H}}$ in $\mathbb{H}$, $e([\mathbf{A}]_{\mathbb{G}}, [\mathbf{B}]_{\mathbb{H}})$ calculates the bilinear maps on each pair of entries of $[\mathbf{A}]_{\mathbb{G}}$ and $[\mathbf{B}]_{\mathbb{H}}$. Formally, $e([\mathbf{A}]_{\mathbb{G}}, [\mathbf{B}]_{\mathbb{H}}) = [\mathbf{A} \otimes \mathbf{B}]_{\mathbb{G}_T}$.

To secure our construction with preprocessing, we need the power DDH assumption to hold in $\mathbb{G}$, and the DDH assumption to hold in $\mathbb{H}$. This assumption has been referred to as *bilinear power DDH* in previous work [CGH⁺21].

Assumption 3 (Bilinear k -Power DDH) *In a given bilinear group $(q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e)$, the k -power DDH assumption holds in $\mathbb{G}$ and the DDH assumption holds in $\mathbb{H}$. That is to say,*

$$(q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e, [\alpha^{-k}x]_{\mathbb{G}}, \dots, [\alpha^{-1}x]_{\mathbb{G}}, [x]_{\mathbb{G}}, [\alpha x]_{\mathbb{G}}, \dots, [\alpha^k x]_{\mathbb{G}}, \\ [y]_{\mathbb{H}}, [z]_{\mathbb{H}}, [yz]_{\mathbb{H}}) \\ \stackrel{c}{\approx} (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e, [r_{-k}]_{\mathbb{G}}, \dots, [r_{-1}]_{\mathbb{G}}, [r_0]_{\mathbb{G}}, [r_1]_{\mathbb{G}}, \dots, [r_k]_{\mathbb{G}}, \\ [y]_{\mathbb{H}}, [z]_{\mathbb{H}}, [r]_{\mathbb{H}})$$

where $x, y, z, \alpha, r, r_{-k}, \dots, r_0, \dots, r_k \xleftarrow{\$} \mathbb{Z}_q^*$.

The intuition behind this assumption is that α, x only appear in $\mathbb{G}$ and y, z only appear in $\mathbb{H}$. Thus, through the pairing, $[y]_{\mathbb{H}}$ and $[z]_{\mathbb{H}}$ cannot help to correlate $[\alpha^i x]_{\mathbb{G}}$ and $[\alpha^j x]_{\mathbb{G}}$. That is, the pairing seems not to weaken the hardness of the power-DDH distinguish game in $\mathbb{G}$. This assumption would hold in the Generic Bilinear Group Model [BB08]. It can be derived from the Schwartz–Zippel lemma that the distinguishing advantage is $O(\frac{k}{q})$ times the square of the number of zero tests.

Our construction without preprocessing relies on the following new assumption. This is, in our view, a fairly natural assumption. It may be used in future works when one tries to cancel out some terms through pairings without sacrificing privacy.

Assumption 4 (Split-Inverse Bilinear DDH) *In the given bilinear group $(q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e)$, it is computationally hard to distinguish $[abc]_{\mathbb{G}_T}$ and $[r]_{\mathbb{G}_T}$ given $q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e, [a, c]_{\mathbb{G}}, [c^{-1}, b]_{\mathbb{H}}$, where $a, b, c, r \xleftarrow{\$} \mathbb{Z}_q^*$.*

The assumption would hold in the Generic Bilinear Group Model [BB08], since abc has degree 3 and all the given information is degree-1. In more detail, the GGM model only allows the distinguisher to compute

$$[\text{linear}(a, c, 1)]_{\mathbb{G}}, [\text{linear}(c^{-1}, b, 1)]_{\mathbb{H}}, [\text{linear}(a, c, c^{-1}, b, ab, ac^{-1}, bc, 1, r)]_{\mathbb{G}_T}$$

(r is replaced by abc in the real world) and zero test them. For any linear function,

$$\begin{aligned} \Pr[r + \text{linear}(a, c, c^{-1}, b, ab, ac^{-1}, bc, 1) = 0] &\leq 1/q, \\ \Pr[abc + \text{linear}(a, c, c^{-1}, b, ab, ac^{-1}, bc, 1) = 0] &\leq 4/q. \end{aligned}$$

The second inequality follows from the Schwartz–Zippel lemma because it is a degree-4 polynomial. Thus, the distinguishing advantage is no more than $\frac{5}{q}$ times the square of the number of zero tests.

When $[c^{-1}]_{\mathbb{H}}$ is not leaked to the distinguisher, this assumption will turn into the conventional BDDH in asymmetric bilinear groups. The name comes from “splitting an inverse of $[c]_{\mathbb{G}}$ into $\mathbb{H}$ ”, because this “split-inverse” term is exactly what complicates the standardization of its security.

2.3 Distributed Discrete Logarithm

Boyle, Gilboa and Ishai [BGI16] proposed a deterministic algorithm to convert subtractive shares into divisional shares. Formally, let $(q, g, \mathbb{G})$ be a cyclic group. Let $\varepsilon > 0$ be the correctness error and $B < q$ be a bound. There exists an algorithm DDLog of running time $O(B/\varepsilon)$, such that, for any $h \in \mathbb{G}$ and $0 \leq x \leq B$,

$$\Pr [\text{DDLog}_{\mathbb{G}, g}(h) - \text{DDLog}_{\mathbb{G}, g}(h \cdot g^x) = x] \geq 1 - \varepsilon$$

where the probability is over a random public function. This is a standard technique in HSS-related constructions. As in other works, we will only use the simplified notation of $\text{DDLog}(\cdot)$.

2.4 Non-Interactive Half-Chosen VOLE (NIHC-VOLE)

In some of the NIHC-VOLE constructions in [ARS24], the setup phase leaves a secret trapdoor to the receiver. We explicitly distinguish whether the receiver gets such a trapdoor, and call the two primitives Non-Interactive Half-Chosen VOLE (NIHC-VOLE) and Non-Interactive Half-Chosen VOLE with Preprocessing (pNIHC-VOLE) respectively.

Definition 1 (Non-Interactive Half-Chosen VOLE). *Let λ be the security parameter and n be the sender's input length. Let $\mathcal{R}$ be a finite ring. A non-interactive half-chosen VOLE scheme over the ring $\mathcal{R}$ consists of five efficient algorithms,*

$$\text{NIHC-VOLE} = (\text{Setup}, \text{Hash}, \text{Enc}, \text{SEval}, \text{REval}),$$

with the following syntax:

- $\text{Setup}(1^\lambda, 1^n) \rightarrow \text{crs}$: Takes as input the security parameter λ and the sender's message length n , outputs a common reference string crs .
- $\text{Hash}(\mathbf{x}, \text{crs}) \rightarrow (\text{hv}, \text{st}_S)$: Take the sender's input $\mathbf{x} \in \mathcal{R}^n$ and the crs as inputs, output the public hash value hv and the private state of sender st_S .
- $\text{Enc}(y, \text{crs}) \rightarrow (\text{ct}, \text{st}_R)$: Take the receiver's input $y \in \mathcal{R}$ and the crs as inputs, output the public ciphertext ct and the private state of receiver st_R .
- $\text{SEval}(\text{st}_S, \text{ct}, \text{crs}) \rightarrow \mathbf{s}_0$: Take as input the sender's private state st_S , the receiver's public ciphertext ct and the crs , output the sender's share $\mathbf{s}_0$.
- $\text{REval}(\text{st}_R, \text{hv}, \text{crs}) \rightarrow \mathbf{s}_1$: Take as input the receiver's private state st_R , the sender's public hash value hv and the crs , output the receiver's share $\mathbf{s}_1$.

The scheme satisfies the following properties:

- ε -Correctness: For all security parameters $\lambda \in \mathbb{N}$ and every $\mathbf{x}, y$, a NIHC-VOLE scheme is said to be ε -correct if

$$\Pr \left[\begin{array}{l} \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^n) \\ (\text{hv}, \text{st}_S) \leftarrow \text{Hash}(\mathbf{x}, \text{crs}) \\ \mathbf{s}_0 - \mathbf{s}_1 = \mathbf{x} \cdot y : (\text{ct}, \text{st}_R) \leftarrow \text{Enc}(y, \text{crs}) \\ \mathbf{s}_0 \leftarrow \text{SEval}(\text{st}_S, \text{ct}, \text{crs}) \\ \mathbf{s}_1 \leftarrow \text{REval}(\text{st}_R, \text{hv}, \text{crs}) \end{array} \right] \geq 1 - \varepsilon.$$

- Sender Privacy: Let $L(\lambda, n)$ denote the length of the randomness needed by $\text{Setup}(1^\lambda, 1^n)$. A NIHC-VOLE scheme satisfies sender privacy if, for every $n \in \mathbb{N}$ and values $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{R}^n$, the following distributions are computationally indistinguishable.

$$\left\{ (\text{hv}, \mathbf{r}) \left| \begin{array}{l} \mathbf{r} \xleftarrow{\$} \{0, 1\}^{L(\lambda, n)} \\ \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^n; \mathbf{r}) \\ (\text{hv}, \text{st}_S) \xleftarrow{\$} \text{Hash}(\mathbf{x}_0, \text{crs}) \end{array} \right. \right\} \left\{ (\text{hv}, \mathbf{r}) \left| \begin{array}{l} \mathbf{r} \xleftarrow{\$} \{0, 1\}^{L(\lambda, n)} \\ \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^n; \mathbf{r}) \\ (\text{hv}, \text{st}_S) \xleftarrow{\$} \text{Hash}(\mathbf{x}_1, \text{crs}) \end{array} \right. \right\}$$

This stronger definition of privacy (even the coins of Setup are public) is called transparent hasher privacy in [ARS24].

- Receiver Privacy: The scheme has receiver privacy if, for every $n \in \mathbb{N}$ and $y_0, y_1 \in \mathcal{R}$, the following distributions are computationally indistinguishable.

$$\left\{ (\text{ct}, \text{crs}) \middle| \begin{array}{l} \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^n) \\ (\text{ct}, \text{st}_R) \leftarrow \text{Enc}(y_0, \text{crs}) \end{array} \right\} \quad \left\{ (\text{ct}, \text{crs}) \middle| \begin{array}{l} \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^n) \\ (\text{ct}, \text{st}_R) \leftarrow \text{Enc}(y_1, \text{crs}) \end{array} \right\}$$

Definition 2 (Non-Interactive Half-Chosen VOLE with Preprocessing).

Let λ be the security parameter and n be the sender's input length. Let $\mathcal{R}$ be a finite ring. An NIHC-VOLE scheme over the ring $\mathcal{R}$ with preprocessing consists of five efficient algorithms,

$$\text{pNIHC-VOLE} = (\text{Prep}, \text{Hash}, \text{Enc}, \text{SEval}, \text{REval}),$$

with the following syntax:

- $\text{Prep}(1^\lambda, 1^n) \rightarrow (\text{prs}, \text{td})$: Takes as input the security parameter λ and the sender's message length n , outputs a public reusable string prs and the private trapdoor td of the receiver.
- $\text{Hash}(\mathbf{x}, \text{prs}) \rightarrow (\text{hv}, \text{st}_S)$: Take the sender's input $\mathbf{x} \in \mathcal{R}^n$ and the prs as inputs, output the public hash value hv and the private state of sender st_S .
- $\text{Enc}(y, \text{td}, \text{prs}) \rightarrow (\text{ct}, \text{st}_R)$: Take the receiver's input $y \in \mathcal{R}$ and the td as inputs, output the public ciphertext ct and the private state of the receiver st_R .
- $\text{SEval}(\text{st}_S, \text{ct}, \text{prs}) \rightarrow \mathbf{s}_0$: Take as input the sender's private state st_S , the receiver's public ciphertext ct and the prs , output the sender's share $\mathbf{s}_0$.
- $\text{REval}(\text{st}_R, \text{hv}, \text{td}, \text{prs}) \rightarrow \mathbf{s}_1$: Take as input the receiver's private state st_R , the sender's public hash value hv and the td , output the receiver's share $\mathbf{s}_1$.

The scheme satisfies the following properties:

- ε -Correctness: For all security parameters $\lambda \in \mathbb{N}$ and every $\mathbf{x}, y$, a pNIHC-VOLE scheme is said to be ε -correct if

$$\Pr \left[\begin{array}{l} (\text{prs}, \text{td}) \leftarrow \text{Prep}(1^\lambda, 1^n) \\ (\text{hv}, \text{st}_S) \leftarrow \text{Hash}(\mathbf{x}, \text{prs}) \\ \mathbf{s}_0 - \mathbf{s}_1 = \mathbf{x} \cdot y : (\text{ct}, \text{st}_R) \leftarrow \text{Enc}(y, \text{td}, \text{prs}) \\ \mathbf{s}_0 \leftarrow \text{SEval}(\text{st}_S, \text{ct}, \text{prs}) \\ \mathbf{s}_1 \leftarrow \text{REval}(\text{st}_R, \text{hv}, \text{td}, \text{prs}) \end{array} \right] \geq 1 - \varepsilon.$$

- Sender Privacy: Let $L(\lambda, n)$ denote the length of the randomness needed by $\text{Setup}(1^\lambda, 1^n)$. The scheme has sender privacy if, for every $n \in \mathbb{N}$ and values $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{R}^n$, the following distributions are computationally indistinguishable.

$$\left\{ (\text{hv}, \mathbf{r}) \middle| \begin{array}{l} \mathbf{r} \xleftarrow{\$} \{0, 1\}^{L(\lambda, n)} \\ (\text{prs}, \text{td}) \leftarrow \text{Prep}(1^\lambda, 1^n; \mathbf{r}) \\ (\text{hv}, \text{st}_S) \xleftarrow{\$} \text{Hash}(\mathbf{x}_0, \text{prs}) \end{array} \right\} \quad \left\{ (\text{hv}, \mathbf{r}) \middle| \begin{array}{l} \mathbf{r} \xleftarrow{\$} \{0, 1\}^{L(\lambda, n)} \\ (\text{prs}, \text{td}) \leftarrow \text{Prep}(1^\lambda, 1^n; \mathbf{r}) \\ (\text{hv}, \text{st}_S) \xleftarrow{\$} \text{Hash}(\mathbf{x}_1, \text{prs}) \end{array} \right\}$$

- Receiver Privacy: Since the receiver holds a trapdoor, the privacy definition has to allow the adversary to access all ciphertexts.
The scheme has receiver privacy if no p.p.t. adversary can win the following game with non-negligible advantage.
Initialisation: Sample $b \leftarrow \{0, 1\}$. Generate $(\text{prs}, \text{td}) \leftarrow \text{Prep}(1^\lambda, 1^n)$.
Provide prs to the adversary.
Query: For $i = 1, \dots, \text{poly}(\lambda)$, receive y_{i0}, y_{i1} from the adversary, compute $(\text{ct}_i, \text{st}_{R,i}) \leftarrow \text{Enc}(y_{ib}, \text{td}, \text{prs})$, provide ct_i to the adversary.
End Game: The adversary wins if it output b .

3 Pairing-based NIHC-VOLE

Our highly succinct NIHC-VOLE is inspired by existing group-based NIHC-VOLE [ARS24]. We briefly recap their construction in Section 3.1, then show how to squeeze the communication using pairings in Section 3.2.

3.1 Warmup Construction

The following construction is an intermediate step adapted from [ARS24]. It should be noted that [ARS24] incorporates additional balancing techniques to achieve sublinear communication.

Construction 1 (DDH-based NIHC-VOLE [ARS24]) *The scheme consists of*

- $\text{Setup}(1^\lambda, 1^n)$: Generate a DDH-hard group as the public parameter $\text{pp} := (q, g, \mathbb{G})$. Sample $a_0, a_1, \dots, a_n \xleftarrow{\$} \mathbb{Z}_q^*$. Output $\text{crs} := (\text{pp}, [a_0]_{\mathbb{G}}, [a_1]_{\mathbb{G}}, \dots, [a_n]_{\mathbb{G}})$.
- $\text{Hash}(\mathbf{x} = (x_1, \dots, x_n), \text{crs})$: Sample $x_0 \xleftarrow{\$} \mathbb{Z}_q$. Output hash value

$$\text{hv} := [a_0]_{\mathbb{G}}^{x_0} [a_1]_{\mathbb{G}}^{x_1} \cdots [a_n]_{\mathbb{G}}^{x_n} = [x_0 a_0 + x_1 a_1 + \cdots + x_n a_n]_{\mathbb{G}}.$$

and sender's private state $\text{st}_S = (x_0, x_1, \dots, x_n)$.

- $\text{Enc}(y, \text{crs})$: Sample $b_1, b_2, \dots, b_n \xleftarrow{\$} \mathbb{Z}_q$. Output ciphertext

$$\text{ct} := \begin{pmatrix} [a_0]_{\mathbb{G}}^{b_1} & [a_0]_{\mathbb{G}}^{b_2} & \cdots & [a_0]_{\mathbb{G}}^{b_n} \\ [a_1]_{\mathbb{G}}^{b_1} [y]_{\mathbb{G}} & [a_1]_{\mathbb{G}}^{b_2} & \cdots & [a_1]_{\mathbb{G}}^{b_n} \\ [a_2]_{\mathbb{G}}^{b_1} & [a_2]_{\mathbb{G}}^{b_2} [y]_{\mathbb{G}} & \cdots & [a_2]_{\mathbb{G}}^{b_n} \\ \vdots & \vdots & \ddots & \vdots \\ [a_n]_{\mathbb{G}}^{b_1} & [a_n]_{\mathbb{G}}^{b_2} & \cdots & [a_n]_{\mathbb{G}}^{b_n} [y]_{\mathbb{G}} \end{pmatrix} = \begin{bmatrix} a_0 b_1 & a_0 b_2 & \cdots & a_0 b_n \\ a_1 b_1 + y & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 + y & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_n + y \end{bmatrix}_{\mathbb{G}}$$

and receiver's private state $\text{st}_R := (b_1, \dots, b_n)$.

- $\text{SEval}(\text{st}_S, \text{ct}, \text{crs})$: Parse the ciphertext ct as a matrix $(\text{ct}_{ij})_{i \in \{0, \dots, n\}, j \in \{1, \dots, n\}}$. Calculate the sender's output share

$$s_S = \text{DDLog}(\text{ct}_{0,1}^{x_0} \text{ct}_{1,1}^{x_1} \cdots \text{ct}_{n,1}^{x_n}, \dots, \text{ct}_{0,n}^{x_0} \text{ct}_{1,n}^{x_1} \cdots \text{ct}_{n,n}^{x_n}).$$

- $\text{REval}(\text{st}_R, \text{hv}, \text{crs})$: Calculate the receiver's output share

$$s_R = \text{DDLog}(\text{hv}^{b_1}, \dots, \text{hv}^{b_n}).$$

Proof of Correctness. We briefly explain the correctness of the above scheme. The j -th coordinate of the sender and the receiver's shares are

$$\begin{aligned} \text{DDLog}(\text{ct}_{0,j}^{x_0} \cdots \text{ct}_{n,j}^{x_n}) &= \text{DDLog}([x_0 a_0 b_j + \cdots + x_n a_n b_j + x_j y]_{\mathbb{G}}), \\ \text{DDLog}(\text{hv}^{b_j}) &= \text{DDLog}([x_0 a_0 b_j + \cdots + x_n a_n b_j]_{\mathbb{G}}). \end{aligned}$$

respectively. And the DDLog algorithm is ε -correct.

The communication of the above construction is highly unbalanced. The sender sends one group element and the receiver sends $O(n^2)$ group elements. As observed by [ARS24], the communication can be balanced by reusing the ciphertext.

Lemma 1. *Given a $\varepsilon(n, \lambda)$ -correct NIHC-VOLE scheme whose CRS, hash value, ciphertext is $\text{cc}_{\text{CRS}}(n, \lambda)$, $\text{cc}_S(n, \lambda)$, $\text{cc}_R(n, \lambda)$ -bit, respectively. Then for any d , there exists a $d\varepsilon(\lceil n/d \rceil, \lambda)$ -correct NIHC-VOLE scheme whose CRS, hash value, and ciphertext are $\text{cc}_{\text{CRS}}(\lceil n/d \rceil, \lambda)$, $d\text{cc}_S(\lceil n/d \rceil, \lambda)$, $\text{cc}_R(\lceil n/d \rceil, \lambda)$ -bit, respectively.*

Proof Sketch. The sender evenly divides its input vector $\mathbf{x}$ into d pieces $\mathbf{x}_1, \dots, \mathbf{x}_d$. They invoke NIHC-VOLE to compute additive shares of $\mathbf{x}_1 \cdot y, \dots, \mathbf{x}_d \cdot y$. In particular, the sender sends a hash value for each of $\mathbf{x}_1, \dots, \mathbf{x}_d$, while the receiver sends only one encryption, which is reused.

3.2 Our Construction based on Split-Inverse BDDH

The communication bottleneck of Construction 1 is the ciphertext matrix. Note that the ciphertext matrix is “close to” low rank. For example, the bottom left of the ciphertext matrix $[(a_{n/2+1}, \dots, a_n)^\top \otimes (b_1, \dots, b_{n/2})]_{\mathbb{G}}$ has rank 1. If we move to group $\mathbb{G}_T$, pairings will provide a succinct representation of this submatrix. The receiver samples a random s and sends $[a_{n/2+1}s^{-1}, \dots, a_ns^{-1}]_{\mathbb{G}}$, $[b_1s, \dots, b_{n/2}s]_{\mathbb{H}}$. The sender can recover the submatrix using the bilinear map

$$\begin{aligned} &[(a_{n/2+1}, \dots, a_n)^\top \otimes (b_1, \dots, b_{n/2})]_{\mathbb{G}_T} \\ &= e([(a_{n/2+1}s^{-1}, \dots, a_ns^{-1})^\top]_{\mathbb{G}}, [(b_1s, \dots, b_{n/2}s)]_{\mathbb{H}}). \end{aligned}$$

Construction 2 *The scheme consists of*

- **Setup**($1^\lambda, 1^n$): *Generate bilinear group* $\text{pp} := (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e)$. *Sample* $a_0, a_1, \dots, a_n \xleftarrow{\$} \mathbb{Z}_q^*$. *Output* $\text{crs} := (\text{pp}, [a_0]_{\mathbb{G}_T}, [a_1]_{\mathbb{G}}, \dots, [a_n]_{\mathbb{G}})$.
- **Hash**($\mathbf{x} = (x_1, \dots, x_n)$, crs): *Sample* $x_0 \xleftarrow{\$} \mathbb{Z}_q$. *Output hash value*

$$\text{hv} := [a_0]_{\mathbb{G}_T}^{x_0} [a_1]_{\mathbb{G}_T}^{x_1} \cdots [a_n]_{\mathbb{G}_T}^{x_n} = [a_0x_0 + \cdots + a_nx_n]_{\mathbb{G}_T}$$

and sender's private state $\text{st}_S = (x_0, x_1, \dots, x_n)$.

- $\text{Enc}(y, \text{crs})$: Sample $b_1, \dots, b_n \xleftarrow{\$} \mathbb{Z}_q^*$. Let $\ell = \lfloor \log n \rfloor + 1$. Sample 2ℓ numbers $s_{1,0}, s_{1,1}, \dots, s_{\ell,0}, s_{\ell,1} \xleftarrow{\$} \mathbb{Z}_q^*$. Let $\text{Bin} : [n+1]^* \rightarrow \{0,1\}^\ell$ be the bit decomposition function, let Bin_k denote the k -th binary bit. Output the ciphertext

$$\begin{aligned} \text{ct} = & \left(e([a_i]_{\mathbb{G}}, [b_i]_{\mathbb{H}}) \cdot [y]_{\mathbb{G}_T} \text{ for } i \in \{1, \dots, n\}, \right. \\ & [a_0]_{\mathbb{G}_T}^{b_j} \text{ for } j \in \{1, \dots, n\}, \\ & [b_j]_{\mathbb{H}}^{s_{k, \text{Bin}_k(j)}} \text{ for } j \in \{1, \dots, n\}, k \in \{1, \dots, \ell\}, \\ & \left. [a_i]_{\mathbb{G}}^{(s_{k, 1 - \text{Bin}_k(i)})^{-1}} \text{ for } i \in \{1, \dots, n\}, k \in \{1, \dots, \ell\} \right). \end{aligned}$$

and the receiver's private state

$$\text{st}_R = (b_1, \dots, b_n, s_{1,0}, s_{1,1}, \dots, s_{\ell,0}, s_{\ell,1}).$$

- $\text{SEval}(\text{st}_S, \text{ct}, \text{crs})$: Compute ciphertext matrix similar to Construction 1

$$\text{ct}' = \begin{pmatrix} [a_0 b_1]_{\mathbb{G}_T} & [a_0 b_2]_{\mathbb{G}_T} & \cdots & [a_0 b_n]_{\mathbb{G}_T} \\ [a_1 b_1 + y]_{\mathbb{G}_T} & [a_1 b_2]_{\mathbb{G}_T} & \cdots & [a_1 b_n]_{\mathbb{G}_T} \\ [a_2 b_1]_{\mathbb{G}_T} & [a_2 b_2 + y]_{\mathbb{G}_T} & \cdots & [a_2 b_n]_{\mathbb{G}_T} \\ \vdots & \vdots & \ddots & \vdots \\ [a_n b_1]_{\mathbb{G}_T} & [a_n b_2]_{\mathbb{G}_T} & \cdots & [a_n b_n + y]_{\mathbb{G}_T} \end{pmatrix}$$

- The first row is given in ct as $[a_0]_{\mathbb{G}_T}^{b_j} = [a_0 b_j]_{\mathbb{G}_T}$.
- The diagonal is also given in ct as $e([a_i]_{\mathbb{G}}, [b_i]_{\mathbb{H}}) \cdot [y]_{\mathbb{G}_T} = [a_i b_i + y]_{\mathbb{G}_T}$.
- The rest of the matrix can be computed using bilinear maps. For any $i \neq j$, there exists k such that i, j differ at the k -th bit, i.e., $\text{Bin}_k(i) \neq \text{Bin}_k(j)$. Find the smallest such k and calculate

$$[a_i b_j]_{\mathbb{G}_T} = e \left([a_i]_{\mathbb{G}}^{(s_{k, 1 - \text{Bin}_k(i)})^{-1}}, [b_j]_{\mathbb{H}}^{s_{k, \text{Bin}_k(j)}} \right).$$

Run the SEval algorithm of Construction 1 on $(\text{ct}', \text{st}_S)$ to get the output s_S .

- $\text{REval}(\text{st}_R, \text{hv}, \text{crs})$: Calculate the receiver's output share

$$s_R = \text{DDLog}(\text{hv}^{b_1}, \dots, \text{hv}^{b_n}).$$

Theorem 5 (NIHC-VOLE). Assuming the Split-Inverse Bilinear DDH assumption and the ε -correctness of DDLog , there exists an NIHC-VOLE scheme with $O(\sqrt{n \log n} \cdot \lambda)$ communication complexity, where n is the input length of the sender. Furthermore, it satisfies $n\varepsilon$ -correctness, sender privacy, and receiver privacy.

Proof of correctness. The same as Construction 1.

Efficiency analysis. The sender sends 1 group element. The receiver sends $O(n \log n)$ group elements. The CRS has $O(n)$ group elements. Using the balancing trick (Lemma 1), both parties send $O(\sqrt{n \log n})$ group elements, and the CRS becomes $O(\sqrt{n / \log n})$ group elements.

Proof of sender privacy. The hash value $\text{hv} = [a_0x_0 + a_1x_1 + \dots + a_nx_n]_{\mathbb{G}_T}$ is one-time padded by sender's private randomness x_0 .

Proof of receiver privacy. We prove by hybrid arguments. Consider hybrid worlds $\text{Hybrid}_0, \dots, \text{Hybrid}_n$. In Hybrid_t ,

$$\begin{aligned} \text{crs} &= (\text{pp}, [a_0]_{\mathbb{G}_T}, [a_1]_{\mathbb{G}} \dots, [a_n]_{\mathbb{G}}), \\ \text{ct} &= \begin{pmatrix} [r_i + y]_{\mathbb{G}_T} \text{ for } i \in \{1, \dots, t\}, \\ [a_i b_i + y]_{\mathbb{G}_T} \text{ for } i \in \{t+1, \dots, n\}, \\ [a_0 b_i]_{\mathbb{G}_T} \text{ for } i \in \{1, \dots, n\}, \\ [a_i]_{\mathbb{G}}^{(s_{k,1} - \text{Bin}_k(i))^{-1}}, [b_i]_{\mathbb{H}}^{s_{k,1} \text{Bin}_k(i)} \text{ for } i \in \{1, \dots, n\}, k \in \{1, \dots, \ell\} \end{pmatrix}. \end{aligned}$$

Hybrid_0 is the real world, in which crs and ct are exactly the same as in Construction 2. In Hybrid_n , the distribution is independent of y , as it is one-time padded by fresh randomness $r_1, \dots, r_n$. Thus receiver privacy holds perfectly in Hybrid_n . It remains to show that Hybrid_{t-1} and Hybrid_t are computationally indistinguishable. Our proof idea is to enhance the distinguisher's ability by leaking additional information within the distinguishing game, and then argue that an adversary fails even in the easier game by reducing it to the game defined in the split-inverse bilinear DDH assumption.

Consider a relaxed task of distinguishing Hybrid_{t-1} from Hybrid_t (which only strengthens the security guarantee). The distinguisher is additionally given y, a_i, b_i, r_i for all $i \neq t$ and $[s_{k,0}^{-1}]_{\mathbb{G}}, [s_{k,1}^{-1}]_{\mathbb{G}}, [s_{k,0}]_{\mathbb{H}}, [s_{k,1}]_{\mathbb{H}}$ for all k . The new distinguishing task is (after ignoring useless terms)

$$\begin{aligned} &\text{distinguish } [a_t b_t]_{\mathbb{G}_T} \text{ and } [r_t]_{\mathbb{G}_T} \\ &\text{given } [a_t]_{\mathbb{G}}, [b_t]_{\mathbb{G}_T}, \\ &\quad [s_{k,0}^{-1}]_{\mathbb{G}}, [s_{k,1}^{-1}]_{\mathbb{G}}, [a_t (s_{k,0})^{-1}]_{\mathbb{G}}, [s_{k,0}]_{\mathbb{H}}, [s_{k,1}]_{\mathbb{H}}, [b_t s_{k,1}]_{\mathbb{H}} \text{ for all } k \end{aligned}$$

Here we assume without loss of generality that $\text{Bin}_k(t) = 1$ for all k .

Further simplify the distinguishing task. The distinguisher is also given $s_{k,0}$ and $s_{k,1}/s_{1,1}$ for all k . The new distinguishing task becomes (after ignoring useless terms)

$$\begin{aligned} &\text{distinguish } [a_t b_t]_{\mathbb{G}_T} \text{ and } [r_t]_{\mathbb{G}_T} \\ &\text{given } [a_t]_{\mathbb{G}}, [s_{1,1}^{-1}]_{\mathbb{G}}, [s_{1,1}]_{\mathbb{H}}, [b_t s_{1,1}]_{\mathbb{H}} \end{aligned}$$

Note that by mapping a, b, c to $a_t, b_t s_{1,1}, s_{1,1}^{-1}$ respectively, this distinguishing game is identical to the game described in Assumption 4. Thus, privacy reduces to the Split-Inverse Bilinear DDH assumption (Assumption 4).

4 Pairing-based NIHC-VOLE with Preprocessing

In this section, we present constructions based on bilinear power DDH. In all constructions, the online communication is $O(1)$ group elements. The first construction in Section 4.1 is based on the bilinear 1-power DDH assumption, and

its public reusable string has $O(n^2)$ group elements. Also in Section 4.1, the second construction reduces the public reusable string to $O(n^{\log_2 3})$ group elements. In Section 4.2, the third construction reduces the public reusable string to $O(n)$ group elements, at the cost of relying on the bilinear n -power DDH assumption.

4.1 Constructions based on Bilinear 1-Power DDH

Construction 3 pNIHC-VOLE = (Prep, Hash, Enc, SEval, REval) as follows.

- **Prep**($1^\lambda, 1^n$): Generate bilinear group $\text{pp} := (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e)$. Sample $a_0, a_1, \dots, a_n, t_0 \xleftarrow{\$} \mathbb{Z}_q^*$. Output trapdoor and public reusable string

$$\begin{aligned} \text{td} &:= (t_0, a_0, a_1, \dots, a_n), \\ \text{prs} &:= \text{pp}, \begin{bmatrix} t_0 a_0 & t_0 a_0 a_1^{-1} & t_0 a_0 a_2^{-1} & \cdots & t_0 a_0 a_n^{-1} \\ t_0 a_1 & \times & t_0 a_1 a_2^{-1} & \cdots & t_0 a_1 a_n^{-1} \\ t_0 a_2 & t_0 a_2 a_1^{-1} & \times & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & t_0 a_{n-1} a_n^{-1} \\ t_0 a_n & t_0 a_n a_1^{-1} & \cdots & t_0 a_n a_{n-1}^{-1} & \times \end{bmatrix}_{\mathbb{G}}. \end{aligned}$$

- **Hash**($\mathbf{x} = (x_1, \dots, x_n), \text{prs}$): Sample $x_0 \xleftarrow{\$} \mathbb{Z}_q$. Take $[t_0 a_0, \dots, t_0 a_n]_{\mathbb{G}}$ from the first column of prs and map them to $\mathbb{G}_T$. Output the hash value and the sender's private state

$$\text{hv} := [t_0 a_0]_{\mathbb{G}_T}^{x_0} \cdot [t_0 a_1]_{\mathbb{G}_T}^{x_1} \cdots [t_0 a_n]_{\mathbb{G}_T}^{x_n}, \quad \text{st}_S := (x_0, x_1, \dots, x_n).$$

- **Enc**(y, td, prs): Sample $t \xleftarrow{\$} \mathbb{Z}_q$. Take the value t_0 from td . Output the ciphertext and the receiver's private state

$$\text{ct} := ([t/t_0]_{\mathbb{H}}, [t + y]_{\mathbb{G}_T}), \quad \text{st}_R := t.$$

- **SEval**($\text{st}_S, \text{ct}, \text{prs}$): Take from prs the following $(n+1) \times n$ matrix (with n blanks)

$$\mathbf{M} := \begin{bmatrix} t_0 a_0 a_1^{-1} & t_0 a_0 a_2^{-1} & \cdots & t_0 a_0 a_n^{-1} \\ \times & t_0 a_1 a_2^{-1} & \cdots & t_0 a_1 a_n^{-1} \\ t_0 a_2 a_1^{-1} & \times & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_0 a_{n-1} a_n^{-1} \\ t_0 a_n a_1^{-1} & \cdots & t_0 a_n a_{n-1}^{-1} & \times \end{bmatrix}_{\mathbb{G}}$$

Calculate $e(\mathbf{M}, [t/t_0]_{\mathbb{H}})$ and then fill all the n blanks with $[t + y]_{\mathbb{G}_T}$. Get a new ciphertext matrix ct' .

$$\text{ct}' = \begin{bmatrix} t a_0 a_1^{-1} & t a_0 a_2^{-1} & \cdots & t a_0 a_n^{-1} \\ t + y & t a_1 a_2^{-1} & \cdots & t a_1 a_n^{-1} \\ t a_2 a_1^{-1} & t + y & \ddots & \vdots \\ \vdots & \ddots & \ddots & t a_{n-1} a_n^{-1} \\ t a_n a_1^{-1} & \cdots & t a_n a_{n-1}^{-1} & t + y \end{bmatrix}_{\mathbb{G}_T}$$

Run the **SEval** algorithm in Construction 1 on $(\text{ct}', \text{st}_S)$, get the sender's share s_S .

- **REval** $(\text{st}_R, \text{hv}, \text{td}, \text{prs})$: Take the values $t_0, a_1, \dots, a_n$ from td , and t from st_R . Calculate and output the receiver's share as

$$s_R = \text{DDLog}(\text{hv}^{a_1^{-1}t/t_0}, \dots, \text{hv}^{a_n^{-1}t/t_0})$$

Proof of correctness. In **SEval**, the sender obtains the ciphertext matrix ct' . Let $b_j = ta_j^{-1}$, then ct' is a direct counterpart of the ciphertext matrix in Construction 1. The correctness proof is the same.

The receiver computes

$$\text{hv}^{a_j^{-1}t/t_0} = [ta_j^{-1}(a_0x_0 + a_1x_1 + \dots + a_nx_n)]_{\mathbb{G}_T}.$$

The sender computes the inner product of st_S and (the j -th column of) ct' ,

$$[ta_0a_j^{-1}x_0 + ta_1a_j^{-1}x_1 + \dots + ta_na_j^{-1}x_n + yx_j]_{\mathbb{G}_T}.$$

They differ by $[x_jy]_{\mathbb{G}_T}$. So **DDLog** outputs an additive share of x_jy with high probability.

Since the following Construction 4 entirely outperforms the above Construction 3 in terms of performance, and moreover, it is based on the same computational assumption, we will directly prove the security of Construction 4.

Construction 3 has very succinct online communication, and size- $O(n^2)$ public reusable string. The next construction reduces the public reusable string length by sampling $a_0, a_1, \dots, a_n$ from a carefully chosen distribution. The idea is to let each a_i be a subset product of a set of randomly sampled values $\{b_1, b_2, \dots, b_{\lceil \log_2(n+2) \rceil}\}$. This will make any cross term a_i/a_j able to be expressed as $\{-1, 0, 1\}$ powers of $\{b_1, b_2, \dots, b_{\lceil \log_2(n+2) \rceil}\}$. For example, let

$$\begin{aligned} a_0 &= b_1, a_1 = b_2, a_2 = b_2b_1, \\ a_3 &= b_3, a_4 = b_3b_1, a_5 = b_3b_2, a_6 = b_3b_2b_1 \dots \end{aligned}$$

In this case, the public reusable string will contain many duplicated entries, e.g., $a_4a_3^{-1} = a_2a_1^{-1}$. So it can be compressed.

Construction 4 (pNIHC-VOLE based on Bilinear 1-Power DDH) *This is the same as Construction 3 except for the distribution of $a_0, \dots, a_n$.*

- **Prep** $(1^\lambda, 1^n)$: Generate $\text{pp} := (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e)$. Let $\ell = \lceil \log_2(n+2) \rceil$. Sample $t_0, b_1, \dots, b_\ell \xleftarrow{\$} \mathbb{Z}_q^*$ and define $a_i := \prod_{j=1}^{\ell} b_j^{\text{Bin}_j(i+1)}$ for each $i \in \{0, 1, \dots, n\}$. Output the trapdoor and the public reusable string

$$\begin{aligned} \text{td} &:= (t_0, b_1, \dots, b_\ell), \\ \text{prs} &:= \text{pp}, \left\{ \left[t_0 \prod_{j=1}^{\ell} b_j^{m_j} \right]_{\mathbb{G}} \mid \begin{array}{l} (m_1, \dots, m_\ell) \in \{-1, 0, 1\}^\ell, \\ (m_1, \dots, m_\ell) \neq \mathbf{0} \end{array} \right\}. \end{aligned}$$

- **Hash**($\mathbf{x} = (x_1, x_2, \dots, x_n), \mathbf{prs}$): Sample $x_0 \xleftarrow{\$} \mathbb{Z}_q$. Take $[t_0 a_0, \dots, t_0 a_n]_{\mathbb{G}}$ from the first column of $\mathbf{prs}$ and map them to $\mathbb{G}_T$. Output the hash value and the sender's private state

$$\mathbf{hv} := [t_0 a_0]_{\mathbb{G}_T}^{x_0} \cdot [t_0 a_1]_{\mathbb{G}_T}^{x_1} \cdots [t_0 a_n]_{\mathbb{G}_T}^{x_n}, \quad \mathbf{st}_S := (x_0, x_1, \dots, x_n).$$

- **Enc**($y, \mathbf{td}, \mathbf{prs}$): Sample $t \xleftarrow{\$} \mathbb{Z}_q$. Take the value t_0 from $\mathbf{td}$. Output the ciphertext and the receiver's private state

$$\mathbf{ct} := ([t/t_0]_{\mathbb{H}}, [t+y]_{\mathbb{G}_T}), \quad \mathbf{st}_R := t.$$

- **SEval**($\mathbf{st}_S, \mathbf{ct}, \mathbf{prs}$): First retrieve from $\mathbf{prs}$ the following matrix of size $(n+1) \times n$ (with n blanks)

$$\mathbf{M} := \begin{bmatrix} t_0 a_0 a_1^{-1} & t_0 a_0 a_2^{-1} & \cdots & t_0 a_0 a_n^{-1} \\ \times & t_0 a_1 a_2^{-1} & \cdots & t_0 a_1 a_n^{-1} \\ t_0 a_2 a_1^{-1} & \times & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_0 a_{n-1} a_n^{-1} \\ t_0 a_n a_1^{-1} & \cdots & t_0 a_n a_{n-1}^{-1} & \times \end{bmatrix}_{\mathbb{G}}$$

Calculate $e(\mathbf{M}, [t/t_0]_{\mathbb{H}})$ and fill all n blanks with $[t+y]_{\mathbb{G}_T}$, get a new ciphertext matrix $\mathbf{ct}'$.

$$\mathbf{ct}' = \begin{bmatrix} t a_0 a_1^{-1} & t a_0 a_2^{-1} & \cdots & t a_0 a_n^{-1} \\ t+y & t a_1 a_2^{-1} & \cdots & t a_1 a_n^{-1} \\ t a_2 a_1^{-1} & t+y & \ddots & \vdots \\ \vdots & \ddots & \ddots & t a_{n-1} a_n^{-1} \\ t a_n a_1^{-1} & \cdots & t a_n a_{n-1}^{-1} & t+y \end{bmatrix}_{\mathbb{G}_T}$$

Run the **SEval** algorithm in Construction 1 on $(\mathbf{ct}', \mathbf{st}_S)$, get the sender's share $\mathbf{s}_S$.

- **REval**($\mathbf{st}_R, \mathbf{hv}, \mathbf{td}, \mathbf{prs}$): Take $t_0, a_1, \dots, a_n$ from $\mathbf{td}$. Take t from $\mathbf{st}_R$. Output the receiver's share as

$$\mathbf{s}_R = \text{DDLog}(\mathbf{hv}^{a_1^{-1}t/t_0}, \dots, \mathbf{hv}^{a_n^{-1}t/t_0})$$

Theorem 6 (pNIHC-VOLE from Bilinear 1-Power DDH). Assuming $\mathbb{G}$ is 1-power DDH-hard, $\mathbb{H}$ is DDH-hard, and **DDLog** is ε -correct, there exists an NIHC-VOLE with preprocessing scheme that satisfies $n\varepsilon$ -correctness, sender privacy, and receiver privacy. The online communication complexity is $O(\lambda)$, which is independent of n , where n is the input length of the sender. The public reusable string has size $O(n^{\log_2 3} \lambda)$.

Proof of correctness. The same as Construction 3.

Proof of sender privacy. The hash value $\mathbf{hv} = [t_0 a_0 x_0 + t_0 a_1 x_1 + \cdots + t_0 a_n x_n]_{\mathbb{G}_T}$ is one-time padded by the sender's private randomness x_0 .

Proof of receiver privacy. We prove by hybrid arguments. In the real world, the adversary receives $\mathbf{prs}$ and L ciphertexts as described in our construction. In the ideal world, the adversary receives entirely random elements. We argue that the real and ideal worlds are computationally indistinguishable. Our proof follows from the observation that generating the ciphertexts essentially relies on $[t_0]_{\mathbb{G}}$. Therefore, by the following lemma, we can replace $(\mathbf{prs}, [t_0]_{\mathbb{G}})$ with uniform random group elements. The lemma is proved later in the section.

Lemma 2. *Assume the 1-power DDH assumption holds for $\mathbb{G}$, then*

$$\{[t_0 b_1^{m_1} \dots b_\ell^{m_\ell}]_{\mathbb{G}} \mid (m_1, \dots, m_\ell) \in \{-1, 0, 1\}^\ell\} \stackrel{c}{\approx} \{[u_1, \dots, u_{3^\ell}]_{\mathbb{G}}\},$$

where $t_0, b_1, \dots, b_\ell, u_1, \dots, u_{3^\ell} \stackrel{s}{\leftarrow} \mathbb{Z}_q^*$.

Consider the adversary's views in the following hybrid worlds.

- **Hybrid₀** (real world): In this world, the adversary receives

$$\mathbf{prs}, [t_1/t_0]_{\mathbb{H}}, [t_1 + y_1]_{\mathbb{G}_T}, \dots, [t_L/t_0]_{\mathbb{H}}, [t_L + y_L]_{\mathbb{G}_T}.$$

Here t_0 denotes the trapdoor head of $\mathbf{prs}$. This is the real world. The adversary receives $\mathbf{prs}$ and L adaptively queried ciphertexts.

- **Hybrid₁**: In this world, the adversary receives

$$\mathbf{prs}, [r_1]_{\mathbb{H}}, [r_1 t_0 + y_1]_{\mathbb{G}_T}, \dots, [r_L]_{\mathbb{H}}, [r_L t_0 + y_L]_{\mathbb{G}_T}.$$

Hybrid₁ is identical to Hybrid₀. Notice that, the ciphertext $([r_i]_{\mathbb{H}}, [r_i t_0 + y_i]_{\mathbb{G}_T})$ can be generated from $[t_0]_{\mathbb{G}}, y_i$ (and r_i is freshly sampled).

- **Hybrid₂**: In this world, the adversary receives

$$\mathbf{prs}', [r_1]_{\mathbb{H}}, [r_1 r_0 + y_1]_{\mathbb{G}_T}, \dots, [r_L]_{\mathbb{H}}, [r_L r_0 + y_L]_{\mathbb{G}_T}.$$

Here r_0 and $\mathbf{prs}'$ consist of uniformly random elements. Lemma 2 shows that it is computationally hard to distinguish $(\mathbf{prs}, [t_0]_{\mathbb{G}})$ and $(\mathbf{prs}', [r_0]_{\mathbb{G}})$. This implies the hardness of distinguishing Hybrid₁ and Hybrid₂. (If there exists an efficient distinguisher that distinguishes between Hybrid₁ and Hybrid₂, there will be an efficient distinguisher that distinguishes between $(\mathbf{prs}, [t_0]_{\mathbb{G}})$ and $(\mathbf{prs}', [r_0]_{\mathbb{G}})$ by sampling r_i, y_i and simulating the views in Hybrid₁ and Hybrid₂. This contradicts Lemma 2.)

- **Hybrid₃**: In this world, the adversary receives

$$\mathbf{prs}', [r_1]_{\mathbb{H}}, [r'_1 + y_1]_{\mathbb{G}_T}, \dots, [r_L]_{\mathbb{H}}, [r'_L + y_L]_{\mathbb{G}_T},$$

This hybrid is indistinguishable from Hybrid₂ since $\mathbb{H}$ is DDH-hard. (Consider an adversary for DDH in $\mathbb{H}$. Upon receiving $([r_0]_{\mathbb{H}}, [r_1]_{\mathbb{H}}, \dots, [r_L]_{\mathbb{H}}, [r_0 r_1]_{\mathbb{H}}, \dots, [r_0 r_L]_{\mathbb{H}})$ or $([r_0]_{\mathbb{H}}, [r_1]_{\mathbb{H}}, \dots, [r_L]_{\mathbb{H}}, [r'_1]_{\mathbb{H}}, \dots, [r'_L]_{\mathbb{H}})$, it simulates the views in Hybrid₂ or Hybrid₃ by sampling $\mathbf{prs}'$, leaving out $[r_0]_{\mathbb{H}}$, and mapping $[r_0 r_i]_{\mathbb{H}}$ or $[r'_i]_{\mathbb{H}}$ to $\mathbb{G}_T$. Thus, an efficient distinguisher between Hybrid₂ and Hybrid₃ implies an efficient distinguisher for the DDH.) This world has perfect privacy since all the L ciphertexts are independent of the plaintexts.

Proof of Lemma 2. Prove by induction. The case $\ell = 0$ trivially holds. For the general case, consider the following indistinguishable distributions.

– The real distribution $\{[t_0 b_1^{m_1} \dots b_\ell^{m_\ell}]_{\mathbb{G}} \mid (m_1, \dots, m_\ell) \in \{-1, 0, 1\}^\ell\}$.

– Hybrid₁ distribution $\begin{bmatrix} r_1, & \dots, & r_{3^{\ell-1}} \\ r_1 b_\ell, & \dots, & r_{3^{\ell-1}} b_\ell \\ r_1 b_\ell^{-1}, & \dots, & r_{3^{\ell-1}} b_\ell^{-1} \end{bmatrix}_{\mathbb{G}}$

By the inductive hypothesis, the real distribution is indistinguishable from the Hybrid₁ distribution.

– Hybrid₂ distribution $\begin{bmatrix} r_1 s_1, & \dots, & r_{3^{\ell-1}} s_1 \\ r_1 s_2, & \dots, & r_{3^{\ell-1}} s_2 \\ r_1 s_3, & \dots, & r_{3^{\ell-1}} s_3 \end{bmatrix}_{\mathbb{G}}$

Hybrid₂ is computationally indistinguishable from Hybrid₁. We additionally give the distinguisher r_i/r_1 , which only strengthens the security guarantee. In this case, an efficient distinguisher between Hybrid₁ and Hybrid₂ implies an efficient distinguisher between $(r_1, r_1 b_\ell, r_1 b_\ell^{-1})$ and uniform, which violates the 1-power DDH assumption.

– Uniform distribution $\{[r_1, \dots, r_{3^\ell}]_{\mathbb{G}}\}$.

It is computationally indistinguishable from Hybrid₂. This follows from the matrix DDH assumption in $\mathbb{G}$, which is equivalent to the DDH assumption.

4.2 Construction based on Bilinear $(n + 1)$ -Power DDH

The next construction is also the same as Construction 3 except for how $a_0, \dots, a_n$ are sampled. Let $a_i = \alpha^{i+1}$, then prs can be compressed to $O(n)$ group elements.

Construction 5

– $\text{Prep}(1^\lambda, 1^n)$: Generate bilinear group $\text{pp} := (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e)$. Sample $\alpha, t_0 \xleftarrow{\$} \mathbb{Z}_q^*$. Output the trapdoor and the public reusable string

$$\begin{aligned} \text{td} &:= \alpha, t_0, \\ \text{prs} &:= \text{pp}, [t_0 \alpha^{-n}]_{\mathbb{G}}, \dots, [t_0 \alpha^{-1}]_{\mathbb{G}}, [t_0 \alpha^1]_{\mathbb{G}}, \dots, [t_0 \alpha^{n+1}]_{\mathbb{G}}. \end{aligned}$$

– $\text{Hash}(\mathbf{x} = (x_1, \dots, x_n), \text{prs})$: Sample $x_0 \xleftarrow{\$} [q]$. Take $[t_0 \alpha^1]_{\mathbb{G}}, \dots, [t_0 \alpha^{n+1}]_{\mathbb{G}}$ from prs and map them to $\mathbb{G}_T$. Output the hash value and the sender's private state

$$\text{hv} = [t_0 \alpha^1]_{\mathbb{G}_T}^{x_0} \cdot [t_0 \alpha^2]_{\mathbb{G}_T}^{x_1} \dots [t_0 \alpha^{n+1}]_{\mathbb{G}_T}^{x_n}, \quad \text{st}_S = (x_0, x_1, \dots, x_n).$$

– $\text{Enc}(y, \text{td}, \text{prs})$: Sample $t \xleftarrow{\$} [q]$. Take t_0 from td . Output the ciphertext and the receiver's private state

$$\text{ct} = ([t/t_0]_{\mathbb{H}}, [t + y]_{\mathbb{G}_T}), \quad \text{st}_R = t.$$

- $\text{SEval}(\text{st}_S, \text{ct}, \text{prs})$: Use ct and prs to calculate the following ciphertext matrix

$$\text{ct}' = \begin{bmatrix} t\alpha^{-1} & t\alpha^{-2} & \cdots & t\alpha^{-n} \\ t+y & t\alpha^{-1} & \cdots & t\alpha^{-n+1} \\ t\alpha^1 & t+y & \ddots & \vdots \\ \vdots & \ddots & \ddots & t\alpha^{-1} \\ t\alpha^{n-1} & \cdots & t\alpha^1 & t+y \end{bmatrix}_{\mathbb{G}_T}.$$

Run the SEval algorithm in Construction 1 on $(\text{ct}', \text{st}_S)$, get the output $\mathbf{s}_S$.

- $\text{REval}(\text{st}_R, \text{hv}, \text{td}, \text{prs})$: Take the value α and t_0 from td . Take the value t from st_R . Calculate and output the receiver's share as

$$\mathbf{s}_R = \text{DDLog}(\text{hv}^{\alpha^{-2}t/t_0}, \dots, \text{hv}^{\alpha^{-n-1}t/t_0})$$

Theorem 7 (pNIHC-VOLE from Bilinear $(n+1)$ -power DDH). *Assuming $\mathbb{G}$ is n -power DDH-hard, $\mathbb{H}$ is DDH-hard, and DDLog is ε -correct, there exists an $n\varepsilon$ -correct pNIHC-VOLE scheme. The online communication complexity is $O(1)$ group elements. The public reusable string has $O(n)$ group elements.*

Proof of correctness. This scheme is a specialized case of Construction 3 with $a_i = \alpha^{i+1}$. Thus its correctness follows from that of Construction 3.

Proof of sender privacy. The hash value $\text{hv} = [t_0\alpha x_0 + t_0\alpha^2 x_1 + \cdots + t_0\alpha^{n+1} x_n]_{\mathbb{G}_T}$ is one-time padded by the sender's private randomness x_0 .

Proof of receiver privacy. Again, we prove by hybrid arguments. The proof is quite similar to that of Construction 4; the only difference is that we reduce to the $(n+1)$ -power DDH instead of 1-power DDH for $\mathbb{G}$.

Consider the adversary's views in the following hybrid worlds.

- Hybrid_0 (real world): In this world, the adversary receives

$$\text{prs}, [t_1/t_0]_{\mathbb{H}}, [t_1 + y_1]_{\mathbb{G}_T}, \dots, [t_L/t_0]_{\mathbb{H}}, [t_L + y_L]_{\mathbb{G}_T}.$$

Here t_0 denotes the trapdoor head of prs . This is the real world, where the adversary receives prs and L adaptively queried ciphertexts.

- Hybrid_1 : In this world, the adversary receives

$$\text{prs}, [r_1]_{\mathbb{H}}, [r_1 t_0 + y_1]_{\mathbb{G}_T}, \dots, [r_L]_{\mathbb{H}}, [r_L t_0 + y_L]_{\mathbb{G}_T}.$$

Hybrid_1 is identical to Hybrid_0 . Notice that, the ciphertext $([r_i]_{\mathbb{H}}, [r_i t_0 + y_i]_{\mathbb{G}_T})$ can be generated from $[t_0]_{\mathbb{G}}, y_i$ (and r_i is freshly sampled).

- Hybrid_2 : In this world, the adversary receives

$$\text{prs}', [r_1]_{\mathbb{H}}, [r_1 r_0 + y_1]_{\mathbb{G}_T}, \dots, [r_L]_{\mathbb{H}}, [r_L r_0 + y_L]_{\mathbb{G}_T}.$$

Here r_0 and prs' consist of uniformly random elements. The $(n+1)$ -power DDH assumption for $\mathbb{G}$ (Assumption 3) implies that it is computationally

hard to distinguish $(\text{prs}, [t_0]_{\mathbb{G}})$ and $(\text{prs}', [r_0]_{\mathbb{G}})$. This implies the hardness of distinguishing Hybrid_1 and Hybrid_2 . (If there exists an efficient distinguisher that distinguishes between Hybrid_1 and Hybrid_2 , there will be an efficient distinguisher that distinguishes between $(\text{prs}, [t_0]_{\mathbb{G}})$ and $(\text{prs}', [r_0]_{\mathbb{G}})$ by sampling r_i, y_i and simulating the views in Hybrid_1 and Hybrid_2 .)

– Hybrid_3 : In this world, the adversary receives

$$\text{prs}', [r_1]_{\mathbb{H}}, [r'_1 + y_1]_{\mathbb{G}_T}, \dots, [r_L]_{\mathbb{H}}, [r'_L + y_L]_{\mathbb{G}_T},$$

This hybrid is indistinguishable from Hybrid_2 since $\mathbb{H}$ is DDH-hard. (Consider an adversary for DDH in $\mathbb{H}$. Upon receiving $([r_0]_{\mathbb{H}}, [r_1]_{\mathbb{H}}, \dots, [r_L]_{\mathbb{H}}, [r_0 r_1]_{\mathbb{H}}, \dots, [r_0 r_L]_{\mathbb{H}})$ or $([r_0]_{\mathbb{H}}, [r_1]_{\mathbb{H}}, \dots, [r_L]_{\mathbb{H}}, [r'_1]_{\mathbb{H}}, \dots, [r'_L]_{\mathbb{H}})$, it simulates the views in Hybrid_2 or Hybrid_3 by sampling prs' , leaving out $[r_0]_{\mathbb{H}}$, and mapping $[r_0 r_i]_{\mathbb{H}}$ or $[r'_i]_{\mathbb{H}}$ to $\mathbb{G}_T$. Thus, an efficient distinguisher between Hybrid_2 and Hybrid_3 implies an efficient distinguisher for the DDH.) This world has perfect privacy since all the L ciphertexts are independent of the plaintexts.

The Insight: Trade-offs between Efficiency and Security. We observe that Construction 4 enjoys a better security guarantee than Construction 5, while suffering from a larger prs . By choosing the distribution of a_i with flexibility, we can actually acquire a series of constructions in the middle. We can relax the $(n+1)$ -power DDH assumption required in Theorem 7 as $(\sqrt[\ell]{n+2} - 1)$ -power DDH, for any positive integer ℓ . The prs size increases by roughly a factor of 2^ℓ (which is constant when ℓ is constant).

In detail, let $\text{Digit} : \{0, \dots, n+1\} \rightarrow \{0, \dots, (\lceil \sqrt[\ell]{n+2} \rceil - 1)\}^\ell$ denote an injective mapping that expresses the input as an ℓ -digit number. Each digit is in the range $\{0, \dots, (\lceil \sqrt[\ell]{n+2} \rceil - 1)\}$. Again, starting from Construction 3, we modify how a_i is sampled in Prep . Sample random values $b_1, \dots, b_\ell$, and let $a_i := \prod_{j=1}^\ell b_j^{\text{Digit}_j(i+1)}$, for $i = 0, 1, \dots, n$. The trapdoor and prs will change correspondingly,

$$\begin{aligned} \text{td} &:= (t_0, b_1, \dots, b_\ell), \\ \text{prs} &:= \text{pp}, \left\{ \left[t_0 \prod_{j=1}^\ell b_j^{m_j} \right]_{\mathbb{G}} \mid \begin{array}{l} (m_1, \dots, m_\ell) \in \{-(\lceil \sqrt[\ell]{n+2} \rceil - 1), \dots, (\lceil \sqrt[\ell]{n+2} \rceil - 1)\}^\ell, \\ (m_1, \dots, m_\ell) \neq \mathbf{0} \end{array} \right\}. \end{aligned}$$

The prs has no more than $2^\ell(n+2)$ group elements.

The security proof is essentially the same, we need an analog of Lemma 2 to show that: assuming $(\sqrt[\ell]{n+2} - 1)$ -power DDH, $(\text{prs}, [t_0]_{\mathbb{G}})$ is indistinguishable from $(\text{prs}', [r]_{\mathbb{G}})$.

References

- AMN⁺18. Nuttapon Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC¹ in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Cham, August 2018.

- AMR25. Damiano Abram, Giulio Malavolta, and Lawrence Roy. Succinct oblivious tensor evaluation and applications: Adaptively-secure laconic function evaluation and trapdoor hashing for all circuits. In Michal Koucký and Nikhil Bansal, editors, *57th ACM STOC*, pages 1875–1886. ACM Press, June 2025.
- ARS24. Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 301–330. Springer, Cham, May 2024.
- BB08. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.
- BBD⁺23. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.
- 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 Boldyreva and Micciancio [BM19], pages 489–518.
- BCG⁺20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Cham, August 2020.
- BCG⁺21. Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In Canteaut and Standaert [CS21], pages 871–900.
- BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.
- BDSS25. Elette Boyle, Lalita Devadas, and Sacha Servan-Schreiber. Non-interactive distributed point functions. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part I*, volume 15674 of *LNCS*, pages 3–35. Springer, Cham, May 2025.
- BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Berlin, Heidelberg, April 2015.
- 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, Berlin, Heidelberg, August 2016.
- BGI17. Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Cham, April / May 2017.

- BGI19. Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 341–371. Springer, Cham, December 2019.
- BM19. Alexandra Boldyreva and Daniele Micciancio, editors. *CRYPTO 2019, Part III*, volume 11694 of *LNCS*. Springer, Cham, August 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, Cham.
- CGH⁺21. Melissa Chase, Sanjam Garg, Mohammad Hajiabadi, Jialin Li, and Peihan Miao. Amortizing rate-1 OT and applications to PIR and PSI. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 126–156. Springer, Cham, November 2021.
- CHHK25. Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. Breaking the $1/\lambda$ -rate barrier for arithmetic garbling. In Fehr and Fouque [FF25], pages 182–213.
- CHP25. Geoffroy Couteau, Aditya Hegde, and Sihang Pu. Enhanced trapdoor hashing from DDH and DCR. In Fehr and Fouque [FF25], pages 123–151.
- CKY25. Geoffroy Couteau, Naman Kumar, and Xiaxi Ye. Multiparty homomorphic secret sharing and more from LPN and MQ. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part II*, volume 16269 of *LNCS*, pages 498–530. Springer, Cham, December 2025.
- CM21. Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Canteaut and Standaert [CS21], pages 842–870.
- CMPR23. Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 194–224. Springer, Cham, April 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, Berlin, 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, Cham, May 2019.
- CS21. Anne Canteaut and François-Xavier Standaert, editors. *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*. Springer, Cham, October 2021.
- DGI⁺19. Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Boldyreva and Micciancio [BM19], pages 3–32.
- DIL⁺20. Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Function secret sharing for PSI-CA: With applications to private contact tracing. Cryptology ePrint Archive, Paper 2020/1599, 2020.
- DIO21. Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPICs*, pages 5:1–5:24. Schloss Dagstuhl, July 2021.

- FF25. Serge Fehr and Pierre-Alain Fouque, editors. *EUROCRYPT 2025, Part VI*, volume 15606 of *LNCS*. Springer, Cham, May 2025.
- GGM24. Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure-aware PSI from incremental function secret sharing. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 309–345. Springer, Cham, August 2024.
- 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, Cham, November 2020.
- GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Berlin, Heidelberg, May 2014.
- GJM03. Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In Matt Blaze, editor, *Financial Cryptography*, pages 120–135, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- GRS22. Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 323–352. Springer, Cham, August 2022.
- GRS23. Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Malicious secure, structure-aware private set intersection. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 577–610. Springer, Cham, August 2023.
- GW25. Aarushi Goel, Mingyuan Wang, and Zhiheng Wang. Multiparty distributed point functions. In Kalai and Kamara [KK25], pages 140–173.
- ILL25a. Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct homomorphic MACs from groups and applications. In *66th FOCS*, pages 2505–2545. IEEE Computer Society Press, December 2025.
- ILL25b. Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. In Kalai and Kamara [KK25], pages 390–425.
- KK25. Yael Tauman Kalai and Seny F. Kamara, editors. *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*. Springer, Cham, August 2025.
- MORS26. Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Computationally succinct authentication from attribute-based laconic function evaluation and more. In *Advances in Cryptology – EUROCRYPT 2026: 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10–14, 2026, Proceedings, Part II*, page 314–344, Berlin, Heidelberg, 2026. Springer-Verlag.
- TSS⁺20. Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy, 2020.
- WYKW21. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.
- 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.

A Applications

A.1 Succinct Homomorphic Secret Sharing

In this section, we formally define succinct HSS with public setup, and show a construction from our NIHC-VOLE. We then establish the formal syntax of our newly proposed succinct HSS with reusable public key, and show a construction from our pNIHC-VOLE. We remind that the settings and definitions of succinct HSS vary in previous works [ARS24, CHP25]. Therefore, we dedicate some discussion to the relationship between them.

Definition 3 (Succinct HSS with Public Setup [CHP25]). *A succinct Homomorphic Secret Sharing (HSS) scheme with public setup for a program class $\mathcal{P}$ is a tuple of algorithms*

$$\text{sHSS} = (\text{HSS.Setup}, \text{HSS.KeyGen}, \text{HSS.Enc}, \text{HSS.Hash}, \text{HSS.Eval}, \text{HSS.HEval})$$

with the following syntax.

- $\text{HSS.Setup}(1^\lambda, 1^n) \rightarrow \text{crs}$ is a PPT setup algorithm that takes as input the security parameter 1^λ and the input length 1^n , and outputs a common reference string crs .
- $\text{HSS.KeyGen}(\text{crs}) \rightarrow (\text{pk}, \text{sk})$ is a PPT setup algorithm that takes crs as input and outputs a pair of public and private keys (pk, sk) .
- $\text{HSS.Enc}(\text{sk}, \mathbf{x}) \rightarrow \text{ct}$ is a PPT algorithm that takes as input the secret key sk and a tuple $\mathbf{x} \in \mathcal{I}^m$, and outputs a ciphertext ct .
- $\text{HSS.Hash}(\text{crs}, \mathbf{y}) \rightarrow (\text{hv}, \psi)$ is a PPT algorithm that takes as input the crs and a tuple $\mathbf{y} \in \mathcal{I}^n$, and outputs a public hash value hv and private input information ψ .
- $\text{HSS.Eval}(\text{sk}, \text{ct}, \text{hv}, P) \rightarrow z_1$ is a polynomial time algorithm that takes as input the secret key sk , ciphertext ct , hash value hv , and a program description $P \in \mathcal{P}$; it outputs a share of the program output z .
- $\text{HSS.HEval}(\text{pk}, \text{ct}, \psi, P) \rightarrow z_0$ is a polynomial time algorithm that takes as input the public key pk , ciphertext ct , private input information ψ , and a program description $P \in \mathcal{P}$; it outputs a share of the program output z .

The scheme should satisfy the following properties:

- **ε -Correctness:** sHSS is ε -correct if for all $\lambda, m, n \in \mathbb{N}$ and all $\mathbf{x} \in \mathcal{I}^m$, $\mathbf{y} \in \mathcal{I}^n$, it holds that

$$\Pr \left[\begin{array}{l} \text{crs} \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{pk}, \text{sk}) \leftarrow \text{HSS.KeyGen}(\text{crs}) \\ \text{ct} \leftarrow \text{HSS.Enc}(\text{sk}, \mathbf{x}) \\ (\text{hv}, \psi) \leftarrow \text{HSS.Hash}(\text{crs}, \mathbf{y}) \\ z_1 \leftarrow \text{HSS.Eval}(\text{sk}, \text{ct}, \text{hv}, P) \\ z_0 \leftarrow \text{HSS.HEval}(\text{pk}, \text{ct}, \psi, P) \end{array} : z_0 - z_1 \neq P(\mathbf{x}, \mathbf{y}) \right] \leq \varepsilon.$$

- **Encryptor Privacy:** sHSS is encryptor-private if for all $\lambda, m, n \in \mathbb{N}$ and inputs $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{I}^m$, the following two distributions are computationally indistinguishable.

$$\left\{ (\text{crs}, \text{pk}, \text{ct}) \left| \begin{array}{l} \text{crs} \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{pk}, \text{sk}) \leftarrow \text{HSS.KeyGen}(\text{crs}) \\ \text{ct} \leftarrow \text{HSS.Enc}(\text{sk}, \mathbf{x}_0) \end{array} \right. \right\}$$

$$\left\{ (\text{crs}, \text{pk}, \text{ct}) \left| \begin{array}{l} \text{crs} \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{pk}, \text{sk}) \leftarrow \text{HSS.KeyGen}(\text{crs}) \\ \text{ct} \leftarrow \text{HSS.Enc}(\text{sk}, \mathbf{x}_1) \end{array} \right. \right\}$$

- **Hasher Privacy:** sHSS is hasher-private if for all $\lambda, m, n \in \mathbb{N}$ and inputs $\mathbf{y}_0, \mathbf{y}_1 \in \mathcal{I}^n$, the following two distributions are computationally indistinguishable.

$$\left\{ (\text{crs}, \text{sk}, \text{hv}) \left| \begin{array}{l} \text{crs} \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{pk}, \text{sk}) \leftarrow \text{HSS.KeyGen}(\text{crs}) \\ (\text{hv}, \psi) \leftarrow \text{HSS.Hash}(\text{crs}, \mathbf{y}_0) \end{array} \right. \right\}$$

$$\left\{ (\text{crs}, \text{sk}, \text{hv}) \left| \begin{array}{l} \text{crs} \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{pk}, \text{sk}) \leftarrow \text{HSS.KeyGen}(\text{crs}) \\ (\text{hv}, \psi) \leftarrow \text{HSS.Hash}(\text{crs}, \mathbf{y}_1) \end{array} \right. \right\}$$

Remark 1. Note that in the original definition of succinct HSS with public setup in [CHP25], the encryption algorithm HSS.Enc takes pk as input, instead of sk . But in all the applications, the party that runs HSS.Enc also runs HSS.Eval , thus holds sk . In our definition above, HSS.Enc essentially runs a private-key encryption instead of a public-key encryption. This modification is mainly for simplicity. The privacy of the hasher is guaranteed by its fresh randomness and is statistically secure.

The public-setup property is critical for [CHP25]’s framework of transforming a succinct HSS into a TDH. Because the crs will serve as the public hash key within the TDH. In contrast, the definition of succinct HSS in [ARS24] does not fit into this framework. However, their construction can be modified to satisfy the definition of [CHP25]. Essentially, the techniques of these two works (which is NIHC-VOLE) are the same.

The following primitive, called algebraic homomorphic message authentication codes (aHMACs), was first proposed in [ILL25a]. We will use it as a gadget in our construction of succinct HSS from NIHC-VOLE. In an aHMAC scheme, a key-tag pair for $x \in \{0, 1\}$ is formulated as (k_x, σ_x) , which satisfies $\sigma_x = k_x + x \cdot \mathbf{s}$, where $\mathbf{s} \in \{0, 1\}^{\ell_s}$ is the global secret key. Both k_x and σ_x lie in $\mathbb{Z}^{\ell_s}$. We use $\sigma_{\mathbf{x}} = (\dots, \sigma_{x[i]}^{(i)}, \dots)$ to denote a vector of such tags for each bit of vector $\mathbf{x}$ (similarly for $k_{\mathbf{x}}$). Roughly speaking, an aHMAC scheme allows two parties, holding the keys $k_{\mathbf{x}}$ and tags $\sigma_{\mathbf{x}}$ respectively, to locally compute $k_{\mathbf{z}}$ and $\sigma_{\mathbf{z}}$ for $\mathbf{z} = C(\mathbf{x})$, for an arbitrary Boolean circuit C .

Definition 4 (Algebraic Homomorphic MACs [ILL25a]). *An algebraic homomorphic MAC scheme aHMAC (for arbitrary Boolean circuits) consists of three efficient algorithms:*

- $\text{KGen}(1^\lambda, \mathbf{s} \in \{0, 1\}^{\ell_s})$ takes a random Boolean vector of global secrets $\mathbf{s}$, and outputs a secret key sk and an evaluation key evk .
- $\text{EvalTag}(\text{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$ takes as inputs the evaluation key evk , a Boolean circuit $C : \{0, 1\}^{\ell_x} \rightarrow \{0, 1\}^{\ell_z}$, input bits $\mathbf{x}$, and their associated tags $\sigma_{\mathbf{x}}$. It outputs tags $\sigma_{\mathbf{z}} \in \mathbb{Z}^{\ell_s \times \ell_z}$ authenticating the outputs of $C(\mathbf{x})$.
- $\text{EvalKey}(\text{sk}, C, k_{\mathbf{x}})$ takes as inputs the secret key sk , a Boolean circuit $C : \{0, 1\}^{\ell_x} \rightarrow \{0, 1\}^{\ell_z}$ and the input keys $k_{\mathbf{x}}$. It outputs MAC keys $k_{\mathbf{z}} \in \mathbb{Z}^{\ell_s \times \ell_z}$ for the outputs of $C(\mathbf{x})$.

As demonstrated in [ILL25a], an aHMAC should satisfy the following properties:

- ε -Correctness: For all $\lambda \in \mathbb{N}$, $\mathbf{s} \xleftarrow{\$} \{0, 1\}^{\ell_s}$, and any input $\mathbf{x}$, it holds that

$$\Pr \left[\begin{array}{l} \sigma_{\mathbf{x}} = k_{\mathbf{x}} + \mathbf{s} \otimes \mathbf{x} \\ (\text{evk}, \text{sk}) \xleftarrow{\$} \text{KGen}(1^\lambda, \mathbf{s}) \\ \sigma_{\mathbf{x}} \neq k_{\mathbf{x}} + \mathbf{s} \otimes \mathbf{x} : \quad \mathbf{x} = C(\mathbf{x}) \\ \sigma_{\mathbf{z}} \leftarrow \text{EvalTag}(\text{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \\ k_{\mathbf{z}} \leftarrow \text{EvalKey}(\text{sk}, C, k_{\mathbf{x}}) \end{array} \right] \leq \varepsilon.$$

- Security: There exists an efficient simulator Sim , such that for any $\lambda, \ell_s \in \mathbb{N}$, any $\mathbf{s} \xleftarrow{\$} \{0, 1\}^{\ell_s}$, and $s = \text{BC}(\mathbf{s})$ the following two distributions are computationally indistinguishable.

$$\{\text{evk} : (\text{evk}, \text{sk}) \xleftarrow{\$} \text{KGen}(1^\lambda, \mathbf{s})\} \quad \{\text{evk} : (\text{evk}, \text{sk}) \xleftarrow{\$} \text{Sim}(1^{\ell_s})\}$$

We have the following constructions of aHMAC:

Lemma 3 (Theorem 5 of [ILL25a]).

1. Assuming circular-power-DDH (CP-DDH) in the non-interactive distributed log sharing (NIDLS) framework (e.g., Damgård-Jurik groups and class groups), or KDM-DCR, there exists an aHMAC scheme achieving negl-correctness.
2. Assuming CP-DDH in prime-order groups, for every polynomial $p(\lambda)$, there exists an aHMAC scheme achieving $1/p$ -correctness.

In the above, evk is of size $\text{poly}(\lambda)$.

Next, we define an extended HSS algorithm ExtEval , first introduced in [CMPR23]. Roughly speaking, it allows an extra multiplication compared to the standard HSS evaluation. We use ExtEval as a subroutine in our construction of succinct HSS. Note that whether the global secret key is formed as a scalar s or a vector $\mathbf{s}$ depends on its instantiation. Specifically, within the DDH-based instantiation [BGI16], the global secret key is a Boolean vector $\mathbf{s}$. This exactly satisfies our needs, because our VOLE only works for small fields.

Lemma 4 (Extended HSS Evaluation). *There exists an efficient algorithm, denoted by ExtEval , that takes as input the shares of x and $\mathbf{s} \cdot x$, denoted as $(\langle x \rangle_0, \langle x \rangle_1)$ and $(\langle x \cdot \mathbf{s} \rangle_0, \langle x \cdot \mathbf{s} \rangle_1)$, and the encryption of $\mathbf{y}$ and $\mathbf{s} \otimes \mathbf{y}$, denoted as $\text{ct}_{\mathbf{y}}$ and $\text{ct}_{\mathbf{s} \otimes \mathbf{y}}$. It outputs a share of $z = x \cdot F(\mathbf{y})$, where F is an RMS program and $\mathbf{s}$ is the global HSS secret key. More formally, if*

$$\begin{aligned} z_0 &\leftarrow \text{ExtEval}(\langle x \rangle_0, \langle x \cdot \mathbf{s} \rangle_0, \text{ct}_{\mathbf{y}}, \text{ct}_{\mathbf{s} \otimes \mathbf{y}}, F), \\ z_1 &\leftarrow \text{ExtEval}(\langle x \rangle_1, \langle x \cdot \mathbf{s} \rangle_1, \text{ct}_{\mathbf{y}}, \text{ct}_{\mathbf{s} \otimes \mathbf{y}}, F), \end{aligned}$$

then

$$\Pr[z_0 - z_1 \neq x \cdot F(\mathbf{y})] \leq \varepsilon,$$

where ε is the correctness error.

Specifically, in the instantiation based on the circular DDH assumption [BGI16], the HSS secret key $\mathbf{s}$ is generated as follows: sample $s \xleftarrow{\$} [q]^*$ where q is the order of the DDH-hard group, and define $\mathbf{s} = \text{Bin}(s)$ ⁸ with $\ell_s := |\mathbf{s}| = \lfloor \log_2 q \rfloor + 1$. The ciphertexts are standard ElGamal encryptions under secret key s . Moreover, F is required to be B -bounded. That is, the magnitude of x , $\mathbf{y}$, and all the intermediate values are bounded by B . For $B \ll q$, the correctness error is inverse-poly.

Our constructions of succinct HSS support evaluating what we call *Bilinear Half-RMS programs*, which slightly extends the function classes of (1) Bilinear RMS programs supported by [CHP25], and (2) Special RMS programs supported by [ARS24]. We formally define it as follows. Note that for the correctness to hold, we conventionally set the bound B to be sufficiently smaller than the group order q .

Definition 5 (Bilinear Half-RMS Program). *A Bilinear Half-RMS program P with bound B takes two inputs $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^*$ and outputs $\mathbf{z} = (z_1, \dots, z_{\ell_{\text{out}}}) \in \mathbb{Z}^{\ell_{\text{out}}}$ such that for each $i \in [1, \ell_{\text{out}}]$, z_i is computed as*

$$z_i = \sum_{j=1}^{\ell} F_{i,j}(\mathbf{x}) \cdot G_{i,j}(\mathbf{y})$$

where $\{G_{i,j}\}_j$ are arbitrary circuits, and $\{F_{i,j}\}_j$ are RMS programs. Furthermore, all the inputs and intermediate values are bounded by B .

Equipped with these preliminaries, we are ready to present the formal construction of our succinct HSS with public setup.

Construction 6 (Succinct HSS for Bilinear Half-RMS Programs) *Given a non-interactive half-chosen VOLE scheme*

$$\text{NIHC-VOLE} = (\text{Setup}, \text{Hash}, \text{Enc}, \text{SEval}, \text{REval}),$$

⁸ We remind that $\text{Bin}(\cdot)$ is the binary decomposition function.

an algebraic homomorphic MAC scheme

$$\text{aHMAC} = (\text{KGen}, \text{EvalTag}, \text{EvalKey}),$$

and an extended HSS evaluation algorithm ExtEval , build a succinct HSS scheme

$$\text{sHSS} = (\text{HSS.Setup}, \text{HSS.KeyGen}, \text{HSS.Enc}, \text{HSS.Hash}, \text{HSS.Eval}, \text{HSS.HEval})$$

for bilinear half-RMS program $P(\mathbf{x}, \mathbf{y}) := F(\mathbf{x}) \cdot G(\mathbf{y})^9$ with $\mathbf{x} \in \{0, 1\}^m$, $\mathbf{y} \in \{0, 1\}^n$, and bound $B \ll q$, as follows:

- $\text{HSS.Setup}(1^\lambda, 1^n)$:
 1. Run the setup algorithm $\text{Setup}(1^\lambda, 1^n)$ of NIHC-VOLE and set its output as crs' .
 2. Let $\text{PRF}_1 : \mathcal{K}_1 \times \{0, 1\}^* \rightarrow [B]$ be a PRF in NC1. Let $\text{PRF}_2 : \mathcal{K}_2 \times \{0, 1\}^* \rightarrow [2^\lambda]$ be a PRF. PRF_2 is used to randomize the shares.
 3. Sample the PRF key $\text{prfk}_2 \xleftarrow{\$} \mathcal{K}_2$.
 4. Output $\text{crs} = (\text{crs}', \text{PRF}_1, \text{PRF}_2, \text{prfk}_2)$.
- $\text{HSS.KeyGen}(\text{crs})$:
 1. Parse the CRS for VOLE crs' from crs .
 2. Sample global secret $s \xleftarrow{\$} [q]^*$, where q is the group order in crs .
 3. For each bit of $\text{Bin}(s)$, run the encryption algorithm Enc of NIHC-VOLE on $(\text{Bin}(s)[i], \text{crs}')$, and get $(\text{ct}_i, \text{st}_R^i)$, for $i \in \{1, \dots, \ell_s\}$.
 4. Sample the PRF key $\text{prfk}_1 \xleftarrow{\$} \mathcal{K}_1$.
 5. Run the KGen algorithm of aHMAC on $(1^\lambda, \text{Bin}(s))$, get (sk', evk) .
 6. Set $\text{pk} = (\text{crs}, \text{ct}_1, \dots, \text{ct}_{\ell_s}, \text{evk})$.
 7. Set $\text{sk} = (\text{crs}, s, \text{st}_R^1, \dots, \text{st}_R^{\ell_s}, \text{prfk}_1, \text{sk}')$.
 8. Output (pk, sk) .
- $\text{HSS.Enc}(\text{sk}, \mathbf{x})$:
 1. Parse crs , s and prfk_1 from sk .
 2. Calculate standard ElGamal encryptions by secret key s , of each bit of prfk_1 and $\text{prfk}_1 \otimes \text{Bin}(s)$, and denote them altogether as $\mathbf{c}_0$.
 3. For $i \in \{1, \dots, m\}$, set $c_i = \text{PRF}_1(\text{prfk}_1, i) + \mathbf{x}[i] \bmod B$.
 4. Output $\text{ct} = (\mathbf{c}_0, c_1, \dots, c_m)$.
- $\text{HSS.Hash}(\text{crs}, \mathbf{y})$:
 1. Parse crs' from crs .
 2. Run the hash algorithm Hash of NIHC-VOLE on $(\mathbf{y}, \text{crs}')$ to get (hv, st_S) .
 3. Set the private input information as $\psi = \text{st}_S$.
 4. Output (hv, ψ) .
- $\text{HSS.Eval}(\text{sk}, \text{ct}, \text{hv}, P)$:
 1. Parse s , st_R^i , crs and sk' from sk .
 2. Parse crs' from crs .
 3. Run the REval algorithm of NIHC-VOLE on $(\text{st}_R^i, \text{hv}, \text{crs}')$ for $i \in \{1, \dots, \ell_s\}$, and get a share of $\text{Bin}(s) \otimes \mathbf{y}$. Denote this share as $\mathbf{s}'_1$.

⁹ F is an RMS program and G is an arbitrary circuit. For clarity of exposition, we consider only this simplified type of bilinear half-RMS program. The extension to multi-output and summation is straightforward.

4. Run the EvalKey algorithm on $(\text{sk}', G, (\mathbf{s}'_1 + (\text{PRF}_2(\text{prfk}_2, (i, j)))_{(i,j) \in [\ell_s] \times [n]}))$ and get a share of $G(\mathbf{y}) \cdot \text{Bin}(s)$, denoted as $k_{G(\mathbf{y})}$.
 5. Parse $c_1, \dots, c_m$ from ct .
 6. Define F' as $F'_{c_1, \dots, c_m}(k) = F((c_j - \text{PRF}_1(k, j) \bmod B)_{j=1}^m)$.
 7. Parse $\mathbf{c}_0$ from ct .
 8. Run the extended HSS evaluation algorithm ExtEval algorithm on $(0, k_{G(\mathbf{y})}, \mathbf{c}_0, F')$ and get z_1 .
 9. Output z_1 .
- HSS.HEval(pk, ct, ψ , P):
1. Parse crs , ct_i and evk from pk .
 2. Parse crs' from crs .
 3. Parse st_S from ψ .
 4. Parse $\mathbf{y}$ from st_S .
 5. Run the SEval algorithm of NIHC-VOLE on $(\text{st}_S, \text{ct}_i, \text{crs}')$ for $i \in \{1, \dots, \ell_s\}$, and get a share of $\text{Bin}(s) \otimes \mathbf{y}$. Denote this share as $\mathbf{s}'_0$.
 6. Run the EvalTag algorithm on $(\text{evk}, G, \mathbf{y}, (\mathbf{s}'_0 + (\text{PRF}_2(\text{prfk}_2, (i, j)))_{(i,j) \in [\ell_s] \times [n]}))$ and get a share of $G(\mathbf{y}) \cdot \text{Bin}(s)$, denoted as $\sigma_{G(\mathbf{y})}$.
 7. Parse $c_1, \dots, c_m$ from ct .
 8. Define F' as $F'_{c_1, \dots, c_m}(k) = F((c_j - \text{PRF}_1(k, j) \bmod B)_{j=1}^m)$.
 9. Parse $\mathbf{c}_0$ from ct .
 10. Run the extended HSS evaluation algorithm ExtEval algorithm on $(G(\mathbf{y}), \sigma_{G(\mathbf{y})}, \mathbf{c}_0, F')$ and get z_0 .
 11. Output z_0 .

Theorem 8 (Succinct HSS with Public Setup for Bilinear Half-RMS Programs). Assume: (1) NIHC-VOLE satisfies ε_1 -correctness, sender privacy and receiver privacy; (2) aHMAC satisfies ε_2 -correctness and security; (3) ExtEval satisfies ε_3 -correctness, the sHSS scheme in Construction 6 is a $(\varepsilon_1 + \varepsilon_2 + \varepsilon_3)\text{poly}(\lambda)$ -correct, hasher-private and encryptor-private succinct HSS scheme with public setup.

Further, if: (1) NIHC-VOLE scheme has crs size L_1 , hash size L_2 and ciphertext size L_3 ; (2) aHMAC scheme has evaluation key size L_4 , then sHSS has crs size $L_1 + \text{poly}(\lambda)$, pk size $L_1 + L_4 + L_3 \text{poly}(\lambda)$, hv size L_2 , ct size $m + \text{poly}(\lambda)$.

Proof. The correctness follows directly from the correctness of NIHC-VOLE, aHMAC and ExtEval of HSS. Hasher-privacy is also easy to argue, since its hash value is one-time padded by its private fresh randomness, and is statistically secure. To prove encryptor-privacy, we construct a sequence of hybrid worlds:

- Hybrid₀: The crs , pk , ct are formulated as in Construction 6. This is the real world.
- Hybrid₁: We change ct_i in pk as outputs of NIHC-VOLE.Enc on 0, instead of $\text{Bin}(s)[i]$. This is indistinguishable from Hybrid₀ by receiver-privacy of NIHC-VOLE. The indistinguishability holds because $\text{Bin}(s)[i]$ serves as messages, which can be chosen by the adversary. Note that in this hybrid, s only appears in evk and $\mathbf{c}_0$.

- Hybrid₂: We change the evaluation key of aHMAC to the output of an efficient simulator $\text{Sim}(1^{\ell_s})$. This is indistinguishable from Hybrid₁ by the privacy of aHMAC, even when g^s is given to the adversary. Note that in this hybrid, s only appears in $\mathbf{c}_0$ as the ElGamal key.
- Hybrid₃: We change $\mathbf{c}_0$ in ct as ElGamal encryption of 0, instead of prfk_1 . This is indistinguishable from Hybrid₂ by the CPA-security of ElGamal. Note that in this hybrid, prfk_1 only appears in $c_1, \dots, c_m$.
- Hybrid₄: We change $c_1, \dots, c_m$ as random values. This is indistinguishable from Hybrid₃ by the security of PRF_1 .

Note that Hybrid₄ is independent of $\mathbf{x}$. Hence the proof is complete.

Specifically, combining the following schemes:

1. NIHC-VOLE scheme in Construction 2 from Split-Inverse BDDH, with $L_1 = O(\sqrt{n/\log n} \cdot \lambda)$, $L_2, L_3 = O(\sqrt{n \log n} \cdot \lambda)$
2. aHMAC scheme in Theorem 3 from various assumptions, with $L_4 = \text{poly}(\lambda)$,

we get an sHSS scheme with crs size $O(\sqrt{n/\log n} \cdot \lambda + \text{poly}(\lambda))$, pk size $O(\sqrt{n \log n} \cdot \text{poly}(\lambda))$, hv size $O(\sqrt{n \log n} \cdot \lambda)$ and ct size $(m + \text{poly}(\lambda))$.

In other words, the communication complexity of sHSS is only $O(\sqrt{n \log n} \cdot \text{poly}(\lambda) + m)$, which is sublinear in one input and rate-1 to the other.

Succinct HSS with public setup for special RMS programs. Note that for special RMS programs, in which G is simply the identity function, we can omit the use of aHMAC in Construction 6. Thus we get the following corollary.

Corollary 1. *Assuming Split-Inverse BDDH, there exists a succinct HSS scheme with public setup for special RMS programs $P = \langle F(\mathbf{x}), \mathbf{y} \rangle$ with bound $B \ll q$. The crs, sk, hv, pk sizes are $O(\sqrt{n \log n} \cdot \text{poly}(\lambda))$, and the ct size is $(m + \text{poly}(\lambda))$.*

Succinct HSS with reusable public key for bilinear half-RMS programs. In this section, we formally introduce our newly proposed primitive — succinct HSS with reusable public key. Compared with the aforementioned succinct HSS with public setup, whose setup algorithm only generates a public CRS, this model's setup algorithm outputs a trapdoor for one of the parties. Further discussion on the syntax is deferred to after the definition.

Definition 6 (Succinct HSS with Reusable Public Key). *A succinct Homomorphic Secret Sharing (HSS) scheme with reusable public key for a program class $\mathcal{P}$ is a tuple of algorithms*

$$\text{sHSS} = (\text{HSS.Setup}, \text{HSS.Enc}, \text{HSS.Hash}, \text{HSS.Eval}, \text{HSS.HEval})$$

with the following syntax.

- $\text{HSS.Setup}(1^\lambda, 1^n) \rightarrow (\text{pk}, \text{td})$ is a PPT setup algorithm that takes as input the security parameter 1^λ and the input length 1^n , outputs a reusable public key pk , and a trapdoor td .

- $\text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \rightarrow (\text{ek}, \text{sk})$ is a PPT setup algorithm that takes as input the security parameter 1^λ , the input length 1^n , the reusable public key pk and the trapdoor td , outputs an evaluation key ek for the hasher, and a secret key sk for the encryptor.
- $\text{HSS.Enc}(\text{pk}, \text{sk}, \mathbf{x}) \rightarrow \text{ct}$ is a PPT algorithm that takes as input the reusable public key pk , the secret key sk and a tuple $\mathbf{x} \in \mathcal{I}^m$, and outputs a ciphertext ct .
- $\text{HSS.Hash}(\text{pk}, \mathbf{y}) \rightarrow (\text{hv}, \psi)$ is a PPT algorithm that takes as input the reusable public key pk and a tuple $\mathbf{y} \in \mathcal{I}^n$, and outputs a public hash value hv and private input information ψ .
- $\text{HSS.Eval}(\text{pk}, \text{sk}, \text{ct}, \text{hv}, P) \rightarrow z_1$ is a polynomial time algorithm that takes as input the pk , the secret key sk , ciphertext ct , hash value hv , and a program description $P \in \mathcal{P}$; it outputs a share of the program output z .
- $\text{HSS.HEval}(\text{pk}, \text{ek}, \text{ct}, \psi, P) \rightarrow z_0$ is a polynomial time algorithm that takes as input the reusable public key pk , ciphertext ct , private input information ψ , and a program description $P \in \mathcal{P}$; it outputs a share of the program output z .

The scheme should satisfy the following properties:

- **ε -Correctness:** sHSS is ε -correct if for all $\lambda, m, n \in \mathbb{N}$ and all $\mathbf{x} \in \mathcal{I}^m$, $\mathbf{y} \in \mathcal{I}^n$, it holds that

$$\Pr \left[\begin{array}{l} (\text{pk}, \text{td}) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{ek}, \text{sk}) \leftarrow \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \\ \text{ct} \leftarrow \text{HSS.Enc}(\text{pk}, \text{sk}, \mathbf{x}) \\ (\text{hv}, \psi) \leftarrow \text{HSS.Hash}(\text{pk}, \mathbf{y}) \\ z_1 \leftarrow \text{HSS.Eval}(\text{pk}, \text{sk}, \text{ct}, \text{hv}, P) \\ z_0 \leftarrow \text{HSS.HEval}(\text{pk}, \text{ek}, \text{ct}, \psi, P) \end{array} : z_0 - z_1 \neq P(\mathbf{x}, \mathbf{y}) \right] \leq \varepsilon.$$

- **Encryptor Privacy:** sHSS is (multi-message) encryptor-private if for all $\lambda, m, n, t \in \mathbb{N}$ and t pairs of inputs $\mathbf{x}_0^1, \mathbf{x}_1^1, \dots, \mathbf{x}_0^t, \mathbf{x}_1^t \in \mathcal{I}^m$, the following two distributions are computationally indistinguishable.

$$\left\{ \text{pk}, \text{ek}, (\text{ct}^i)_{i=1}^t \mid \begin{array}{l} (\text{pk}, \text{td}) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{ek}, \text{sk}) \leftarrow \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \\ \text{ct}^i \leftarrow \text{HSS.Enc}(\text{pk}, \text{sk}, \mathbf{x}_0^i) \text{ for } i \in \{1, \dots, t\} \end{array} \right\}$$

$$\left\{ \text{pk}, \text{ek}, (\text{ct}^i)_{i=1}^t \mid \begin{array}{l} (\text{pk}, \text{td}) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{ek}, \text{sk}) \leftarrow \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \\ \text{ct}^i \leftarrow \text{HSS.Enc}(\text{pk}, \text{sk}, \mathbf{x}_1^i) \text{ for } i \in \{1, \dots, t\} \end{array} \right\}$$

- **Hasher Privacy:** sHSS is hasher-private if for all $\lambda, m, n \in \mathbb{N}$ and inputs $\mathbf{y}_0, \mathbf{y}_1 \in \mathcal{I}^n$, the following two distributions are computationally indistin-

guishable.

$$\left\{ \begin{array}{l} (\text{pk}, \text{td}) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{pk}, \text{sk}, \text{hv}) \left| \begin{array}{l} (\text{ek}, \text{sk}) \leftarrow \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \\ (\text{hv}, \psi) \leftarrow \text{HSS.Hash}(\text{pk}, \mathbf{y}_0) \end{array} \right. \end{array} \right\}$$

$$\left\{ \begin{array}{l} (\text{pk}, \text{td}) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ (\text{pk}, \text{sk}, \text{hv}) \left| \begin{array}{l} (\text{ek}, \text{sk}) \leftarrow \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \\ (\text{hv}, \psi) \leftarrow \text{HSS.Hash}(\text{pk}, \mathbf{y}_1) \end{array} \right. \end{array} \right\}$$

- **Public Key Reusability:** sHSS is public key reusable, if for all $\lambda, m, n, \ell \in \mathbb{N}$ and ℓ pairs of inputs $\mathbf{x}_0^1, \mathbf{x}_1^1, \dots, \mathbf{x}_0^\ell, \mathbf{x}_1^\ell \in \mathcal{I}^m$, the following two distributions are computationally indistinguishable.

$$\left\{ \begin{array}{l} (\text{pk}, \text{td}) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ \text{pk}, (\text{ek}_i, \text{ct}^i)_{i=1}^\ell \left| \begin{array}{l} (\text{ek}_i, \text{sk}_i) \xleftarrow{\$} \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \text{ for } i \in \{1, \dots, \ell\} \\ \text{ct}^i \leftarrow \text{HSS.Enc}(\text{pk}, \text{sk}_i, \mathbf{x}_0^i) \text{ for } i \in \{1, \dots, \ell\} \end{array} \right. \end{array} \right\}$$

$$\left\{ \begin{array}{l} (\text{pk}, \text{td}) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n) \\ \text{pk}, (\text{ek}_i, \text{ct}^i)_{i=1}^\ell \left| \begin{array}{l} (\text{ek}_i, \text{sk}_i) \xleftarrow{\$} \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td}) \text{ for } i \in \{1, \dots, \ell\} \\ \text{ct}^i \leftarrow \text{HSS.Enc}(\text{pk}, \text{sk}_i, \mathbf{x}_1^i) \text{ for } i \in \{1, \dots, \ell\} \end{array} \right. \end{array} \right\}$$

Remark 2. The definition of succinct HSS in [ARS24] is almost identical to ours, ignoring the public key reusability property. In their definition, the setup phase outputs a public key pk and a pair of evaluation keys $(\text{ek}_0, \text{ek}_1)$ for each party. There are mainly two differences: (1) we use a secret-key encryption in HSS.Enc , while they use a public-key one; (2) they wrap HSS.Setup and HSS.KeyGen into a single setup algorithm.

Looking closer, their definition is essentially a bilateral analogue of ours. Suppose we run

$$(\text{pk}_A, \text{td}_A) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n), (\text{ek}_A, \text{sk}_A) \leftarrow \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}_A, \text{td}_A)$$

for Alice and

$$(\text{pk}_B, \text{td}_B) \leftarrow \text{HSS.Setup}(1^\lambda, 1^n), (\text{ek}_B, \text{sk}_B) \leftarrow \text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}_B, \text{td}_B)$$

for Bob. Set $\text{pk} = (\text{pk}_A, \text{pk}_B)$, $\text{ek}_0 = (\text{ek}_B, \text{sk}_A)$ and $\text{ek}_1 = (\text{ek}_A, \text{sk}_B)$. Then, defining the above altogether as a setup algorithm matches their syntax.

In their setting, the two parties hold shares (s_0, s_1) of the HSS secret key s , while in our setting s is held by one party. They employed this model to allow both parties to have special inputs, $\mathbf{x}_0$ and $\mathbf{x}_1$ respectively¹⁰. As for

¹⁰ In a special RMS program, special inputs refer to those inputs on which the output depends linearly. They are also the inputs that can be compressed in the communication.

construction, they call two NIHC-VOLEs to obtain shares of $s_0 \cdot \mathbf{x}_1$ and $s_1 \cdot \mathbf{x}_0$. Then they locally transfer them to shares of $s \cdot \mathbf{x}_1$ and $s \cdot \mathbf{x}_0$. In contrast, in our definition, one party knows s , and only the other party has the special input $\mathbf{x}$. They call one NIHC-VOLE to obtain shares of $s \cdot \mathbf{x}$. We note that our unilateral definition already suffices for most applications of succinct HSS.

By combining our pNIHC-VOLE scheme with aHMAC and extended HSS evaluation, we acquire the following construction for succinct HSS with reusable public key.

Construction 7 (Succinct HSS with Reusable Public Key) *Given an NIHC-VOLE with preprocessing scheme*

$$\text{pNIHC-VOLE} = (\text{Prep}, \text{Hash}, \text{Enc}, \text{SEval}, \text{REval}),$$

an algebraic homomorphic MAC scheme

$$\text{aHMAC} = (\text{KGen}, \text{EvalTag}, \text{EvalKey}),$$

and an extended HSS evaluation algorithm ExtEval, build a succinct HSS scheme

$$\text{sHSS} = (\text{HSS.Setup}, \text{HSS.KeyGen}, \text{HSS.Enc}, \text{HSS.Hash}, \text{HSS.Eval}, \text{HSS.HEval})$$

for bilinear half-RMS program $P(\mathbf{x}, \mathbf{y}) := F(\mathbf{x}) \cdot G(\mathbf{y})$ with $\mathbf{x} \in \{0, 1\}^m$, $\mathbf{y} \in \{0, 1\}^n$ and bound $B \ll q$, as follows:

- $\text{HSS.Setup}(1^\lambda, 1^n)$:
 1. Run the preprocessing algorithm $\text{Prep}(1^\lambda, 1^n)$ of pNIHC-VOLE and set its output as (prs, td) .
 2. Let $\text{PRF}_1 : \mathcal{K}_1 \times \{0, 1\}^* \rightarrow B$ be a PRF. Let $\text{PRF}_2 : \mathcal{K}_2 \times \{0, 1\}^* \rightarrow [2^\lambda]$ be a PRF. PRF_2 is used to randomize the shares.
 3. Set $\text{pk} = (\text{prs}, \text{PRF}_1, \text{PRF}_2)$.
 4. Output (pk, td) .
- $\text{HSS.KeyGen}(1^\lambda, 1^n, \text{pk}, \text{td})$:
 1. Parse $\text{prs}, \text{PRF}_1, \text{PRF}_2$ from pk .
 2. Sample a PRF key $\text{prfk}_1 \xleftarrow{\$} \mathcal{K}_1$. Sample a PRF key $\text{prfk}_2 \xleftarrow{\$} \mathcal{K}_2$.
 3. Sample a global secret $s \xleftarrow{\$} [q]^*$, where q is the group order in prs .
 4. For each bit of $\text{Bin}(s)$, run the encryption algorithm Enc of pNIHC-VOLE on $(\text{Bin}(s)[i], \text{td}, \text{prs})$, and get $(\text{ct}_i, \text{st}_R^i)$, for $i \in \{1, \dots, \ell_s\}$.
 5. Run the KGen algorithm of aHMAC on $(1^\lambda, \text{Bin}(s))$, get (sk', evk) .
 6. Set $\text{ek} = (\text{ct}_1, \dots, \text{ct}_{\ell_s}, \text{prfk}_2, \text{evk})$.
 7. Set $\text{sk} = (s, \text{td}, \text{st}_R^1, \dots, \text{st}_R^{\ell_s}, \text{prfk}_1, \text{prfk}_2, \text{sk}')$.
 8. Output (ek, sk) .
- $\text{HSS.Enc}(\text{pk}, \text{sk}, \mathbf{x})$:
 1. Parse prs and PRF_1 from pk ¹¹.
 2. Parse s and prfk_1 from sk .

¹¹ We only need the group parameters pp in prs to do encryption, not all of prs . However, we take the whole pk as input to Enc for convenience.

3. Calculate standard ElGamal encryptions by secret key s , of each bit of prfk_1 and $\text{prfk}_1 \otimes \text{Bin}(s)$, and denote them altogether as $\mathbf{c}_0$.
 4. For $i \in \{1, \dots, m\}$, set $c_i = \text{PRF}_1(\text{prfk}_1, i) + \mathbf{x}[i] \bmod B$.
 5. Output $\text{ct} = (\mathbf{c}_0, c_1, \dots, c_m)$.
- $\text{HSS.Hash}(\text{pk}, \mathbf{y})$:
1. Parse prs from pk .
 2. Run the hash algorithm Hash of pNIHC-VOLE on $(\mathbf{y}, \text{prs})$ to get (hv, st_S) .
 3. Set the private input information as $\psi = \text{st}_S$.
 4. Output (hv, ψ) .
- $\text{HSS.Eval}(\text{pk}, \text{sk}, \text{ct}, \text{hv}, P)$:
1. Parse $\text{prs}, \text{PRF}_1, \text{PRF}_2$ from pk ¹².
 2. Parse $s, \text{st}_R^i, \text{td}, \text{prfk}_2$ and sk' from sk .
 3. Run the REval algorithm of pNIHC-VOLE on $(\text{st}_R^i, \text{hv}, \text{td}, \text{prs})$ for $i \in \{1, \dots, \ell_s\}$, and get a share of $\text{Bin}(s) \otimes \mathbf{y}$. Denote this share as $\mathbf{s}'_1$.
 4. Run the EvalKey algorithm on $(\text{sk}', G, (\mathbf{s}'_1 + (\text{PRF}_2(\text{prfk}_2, (i, j)))_{(i,j) \in [\ell_s] \times [n]}))$ and get a share of $G(\mathbf{y}) \cdot \text{Bin}(s)$, denoted as $k_{G(\mathbf{y})}$.
 5. Parse $c_1, \dots, c_m$ from ct .
 6. Define F' as $F'_{c_1, \dots, c_m}(k) = F((c_j - \text{PRF}_1(k, j) \bmod B)_{j=1}^m)$.
 7. Parse $\mathbf{c}_0$ from ct .
 8. Run the extended HSS evaluation algorithm ExtEval algorithm on $(0, k_{G(\mathbf{y})}, \mathbf{c}_0, F')$ and get z_1 .
 9. Output z_1 .
- $\text{HSS.HEval}(\text{pk}, \text{ek}, \text{ct}, \psi, P)$:
1. Parse prs, PRF_1 and PRF_2 from pk .
 2. Parse $\text{ct}_i, \text{prfk}_2$ and evk from ek .
 3. Parse st_S from ψ . Parse $\mathbf{y}$ from st_S .
 4. Run the SEval algorithm of pNIHC-VOLE on $(\text{st}_S, \text{ct}_i, \text{prs})$ for $i \in \{1, \dots, \ell_s\}$, and get a share of $\text{Bin}(s) \otimes \mathbf{y}$. Denote this share as $\mathbf{s}'_0$.
 5. Run the EvalTag algorithm on $(\text{evk}, G, \mathbf{y}, (\mathbf{s}'_0 + (\text{PRF}_2(\text{prfk}_2, (i, j)))_{(i,j) \in [\ell_s] \times [n]}))$ and get a share of $G(\mathbf{y}) \cdot \text{Bin}(s)$, denoted as $\sigma_{G(\mathbf{y})}$.
 6. Parse $c_1, \dots, c_m$ from ct .
 7. Define F' as $F'_{c_1, \dots, c_m}(k) = F((c_j - \text{PRF}_1(k, j) \bmod B)_{j=1}^m)$.
 8. Parse $\mathbf{c}_0$ from ct .
 9. Run the extended HSS evaluation algorithm ExtEval algorithm on $(G(\mathbf{y}), \sigma_{G(\mathbf{y})}, \mathbf{c}_0, F')$ and get z_0 .
 10. Output z_0 .

Remark 3. Notice that in our construction, ek, sk and $\mathbf{c}_0$ are independent of the messages $\mathbf{x}$ and $\mathbf{y}$. Hence they are reusable for different inputs. However, they depend on the HSS secret key s . The public key, in contrast, is reusable even for different choices of s . Its function is more akin to a CRS.

¹² We only use pp and these PRFs in pk for encryptor's evaluation. We take the whole pk as input for convenience.

Theorem 9 (Succinct HSS with Reusable Public Key for Bilinear Half-RMS Programs). *Assume: (1) pNIHC-VOLE satisfies ε_1 -correctness, sender privacy and receiver privacy; (2) aHMAC satisfies ε_2 -correctness and security; (3) ExtEval satisfies ε_3 -correctness, the sHSS scheme in Construction 7 is a $(\varepsilon_1 + \varepsilon_2 + \varepsilon_3) \text{poly}(\lambda)$ -correct, hasher-private and encryptor-private succinct HSS scheme with reusable public key.*

Further, if: (1) pNIHC-VOLE scheme has public reusable string size L_1 , trap-door size L_2 , hash size L_3 , ciphertext size L_4 , receiver private state size L_5 ; (2) aHMAC scheme has evaluation key size L_6 , secret key size L_7 , then sHSS has pk size $L_1 + \text{poly}(\lambda)$, ek size $L_4 \text{poly}(\lambda) + L_6 + \lambda$, sk size $L_2 + L_5 \text{poly}(\lambda) + L_7 + 2\lambda$, hv size L_3 , ct size $m + \text{poly}(\lambda)$.

Proof. The proof is almost the same as that of Theorem 8. The correctness follows directly from the correctness of pNIHC-VOLE, aHMAC and ExtEval of HSS. Hasher-privacy also holds statistically.

To prove encryptor-privacy, we also construct a sequence of hybrid worlds:

- Hybrid₀: The pk, ek, ct^{*i*} are formulated as in Construction 7. This is the real world.
- Hybrid₁: We change ct_{*j*} in ek to outputs of pNIHC-VOLE.Enc on 0, instead of Bin(*s*)[*j*]. This is indistinguishable from Hybrid₀ by (multi-message) receiver-privacy of pNIHC-VOLE.
- Hybrid₂: We change the evaluation key of aHMAC to the output of an efficient simulator Sim(1^{ℓ_s}). This is indistinguishable from Hybrid₁ by the privacy of aHMAC. Now *s* only appears in **c**₀ as the ElGamal key.
- Hybrid₃: We change **c**₀ in ct^{*i*} to ElGamal encryption of 0, instead of prfk₁. This is indistinguishable from Hybrid₂ by CPA-security of ElGamal.
- Hybrid₄: We change all the $c_1^i, \dots, c_m^i$ in ct^{*i*} to random values. This is indistinguishable from Hybrid₃ by security of PRF₁.

Note that Hybrid₄ is independent of $\mathbf{x}^i$. Hence the proof is complete.

The proof of public key reusability is very similar to that of encryptor-privacy. The only difference is that we rely on the multi-message privacy of the underlying pNIHC-VOLE. We construct a sequence of hybrid worlds:

- Hybrid₀: This is the real world.
- Hybrid₁: Change all the ct_{*j*}^{*i*} in ek_{*i*}, for $j = 1, \dots, \ell_s$ and $i = 1, \dots, \ell$, to pNIHC-VOLE.Enc of 0. This is indistinguishable from Hybrid₀ by (multi-message) receiver-privacy of pNIHC-VOLE.
- Hybrid₂: Change all the aHMAC evaluation keys in ek₁, ..., ek _{ℓ} to outputs of efficient simulators. This is indistinguishable from Hybrid₀ by the (multi-message, adaptive) privacy of aHMAC.
- Hybrid₃: Change **c**₀^{*i*} in ct^{*i*} to ElGamal encryption of 0, instead of prfk₁^{*i*}, for $i = 1, \dots, \ell$. This is indistinguishable from Hybrid₂ by CPA-security of ElGamal. Now the PRF keys only appear in $c_1^i, \dots, c_m^i$.
- Hybrid₄: Change all the $c_1^i, \dots, c_m^i$ in ct^{*i*} to random values. This is indistinguishable from Hybrid₃ by the security of PRF.

Note that Hybrid_4 is independent of $\mathbf{x}^i$. Hence the proof is complete.

Specifically, combining the following schemes:

1. **pNIHC-VOLE** scheme in Construction 5 from bilinear power-DDH, with $L_1 = O(n\lambda)$, $L_2, L_3, L_4, L_5 = O(\lambda)$,
2. **aHMAC** scheme in Theorem 3 from various assumptions, with $L_6 = \text{poly}(\lambda)$, $L_7 = \text{poly}(\lambda)$,

we obtain a **sHSS** scheme with reusable **pk** size $O(n\lambda) + \text{poly}(\lambda)$, **ek** and **sk** size $\text{poly}(\lambda)$, **hv** size $O(\lambda)$ and **ct** size $m + \text{poly}(\lambda)$. In other words, with a $(O(n\lambda) + \text{poly}(\lambda))$ -size reusable public key, the communication complexity of **sHSS** is only $m + \text{poly}(\lambda)$, which is independent of one input and rate-1 to the other.

Succinct HSS with reusable public key for special RMS programs. Note that for special RMS programs, in which G is simply an identity function, we can omit the use of **aHMAC** in Construction 7. Thus we get the following corollary.

Corollary 2. *Assuming bilinear power-DDH, there exists a succinct HSS scheme with reusable public key for special RMS programs $P = \langle F(\mathbf{x}), \mathbf{y} \rangle$ with bound $B \ll q$. The **pk** size is $O(n\lambda) + \text{poly}(\lambda)$. The **ek** and **sk** sizes are $\text{poly}(\lambda)$. The **hv** size is $O(\lambda)$. The **ct** size is $m + \text{poly}(\lambda)$.*

A.2 Multiparty Distributed Point Functions

Definition 7 (N -party DPF against $(N - 1)$ -Corruptions). *An N -party distributed point function scheme over $\mathcal{R}$ against $(N - 1)$ static, semi-honest corruptions consists of two efficient algorithms:*

- **Gen**: takes as input the security parameter λ , domain size 2^k , $\alpha \in [2^k]$ and $\beta \in \mathcal{R}$. It outputs keys $\kappa_1, \dots, \kappa_N$.
- **Eval**: takes as input a key κ_i . It outputs a vector $\mathbf{s}_i \in \mathcal{R}^{2^k}$.

It should satisfy two properties:

- *Correctness*: for any $\ell \in [2^k]$:

$$\sum_{i=1}^N \mathbf{s}_i[\ell + 1] = \begin{cases} \beta & \text{if } \ell = \alpha, \\ 0 & \text{else.} \end{cases}$$

- *Privacy*: for any α, β and any $i \in \{1, \dots, N\}$:

$$\begin{aligned} & \{(\alpha, \beta), \kappa_j \text{ for all } j \neq i : (\kappa_1, \dots, \kappa_N) \leftarrow \text{Gen}(1^\lambda, 2^k, \alpha, \beta)\} \\ & \stackrel{c}{\approx} \{(r_1 \xleftarrow{\$} [2^k], r_2 \xleftarrow{\$} \mathcal{R}), \kappa_j \text{ for all } j \neq i : (\kappa_1, \dots, \kappa_N) \leftarrow \text{Gen}(1^\lambda, 2^k, \alpha, \beta)\}. \end{aligned}$$

Remark 4. We define the evaluation algorithm **Eval** directly as a full-domain evaluation. This suffices for almost all the applications of DPF.

As described in the introduction, [ARS24] has proposed a complete framework that transforms a succinct HSS (with public setup) scheme for special RMS programs into an N -party distributed point function (DPF). We omit the detailed constructions and proofs, presenting only the main result.

Lemma 5 (Theorem 12 of [ARS24], Rephrased). *Assume there exists a succinct HSS scheme (with public setup) for special RMS programs over $\mathbb{Z}_2$ ¹³, then there exists an N -party DPF scheme for the domain $[2^k]$ against static, semi-honest adversaries corrupting up to $(N - 1)$ parties. Further, assume the HSS scheme has crs size T_1 , (pk, sk) size T_2 , hv size T_3 , then the DPF scheme has key size $Nk \text{poly}(\lambda) + O(N)(T_1 + T_2 + T_3)$.*

Combining the above lemma with our succinct HSS for special RMS programs in Corollary 1, we get the following theorem.

Theorem 10 (N -party DPF, Improved). *Assuming the Split-Inverse BDDH assumption, there exists an N -party DPF on domain $[2^k]$ and ring $\mathbb{Z}_2$, against static, semi-honest adversaries corrupting up to $N - 1$ parties with key size $O(N \cdot 2^{k/2} \cdot \sqrt{k} \cdot \text{poly}(\lambda))$.*

Multi-Party DPF in the CRS Model. We now consider a modified model of multi-party DPF, which we call MPDPF in the CRS model. In this model, the dealer generates a relatively large (structured) CRS in the setup phase, which is public. Moreover, the DPF key generated for each party consists of two parts:

1. a succinct reusable head, which is private to the party;
2. a super-succinct fresh key generated per instantiation, which is also private to the party.

This model, with super-succinct communication after amortization, may be of independent interest.

Definition 8 (N -party DPF in the CRS Model against $(N - 1)$ Corruptions). *An N -party distributed point function scheme over a ring $\mathcal{R}$, against $(N - 1)$ static, semi-honest corruptions in the CRS model consists of three efficient algorithms:*

- **Setup:** takes as input the security parameter λ and the domain size 2^k . It outputs a structured crs and a private state st .
- **Gen:** takes as input crs , st , $\alpha \in [2^k]$ and $\beta \in \mathcal{R}$. It outputs keys $\kappa_1, \dots, \kappa_N$. Each κ_i consists of two parts, the reusable key κ_i^{reuse} and the fresh key κ_i^{fresh} .
- **Eval:** takes as input the crs , and a key κ_i . It outputs a vector $\mathbf{s}_i \in \mathcal{R}^{2^k}$, which is party- i 's share of the full domain evaluation of DPF.

It should satisfy two properties:

¹³ Recap that our succinct HSS works for B -bounded programs where $B \ll q$. Boolean programs over $\mathbb{Z}_2$ are naturally 2-bounded.

- *Correctness*: for any $\ell \in [2^k]$:

$$\sum_{i=1}^N s_i[\ell + 1] = \begin{cases} \beta & \text{if } \ell = \alpha, \\ 0 & \text{else.} \end{cases}$$

- *Privacy*: For any subset I of $\{1, \dots, N\}$ of size $N - 1$, any $t \in \mathbb{N}$ and any $\alpha_j \in [2^k]$, $\beta_j \in \mathcal{R}$ for $j = 1, \dots, t$, the following two distributions are computationally indistinguishable.

$$\left\{ \begin{array}{l} \alpha_j, \beta_j, \text{crs}, \kappa_i^{\text{reuse}}, (\kappa_i^{\text{fresh}})^j \\ \text{for } j \in \{1, \dots, t\} \text{ and } i \in I \end{array} \middle| \begin{array}{l} (\text{crs}, \text{st}) \leftarrow \text{Setup}(1^\lambda, 2^k) \\ (\kappa_i^{\text{reuse}} || (\kappa_i^{\text{fresh}})^j) \leftarrow \text{Gen}(\text{crs}, \text{st}, \alpha_j, \beta_j) \\ \text{for } j \in \{1, \dots, t\} \text{ and } i \in \{1, \dots, N\} \end{array} \right\}$$

$$\left\{ \begin{array}{l} r_j \xleftarrow{\$} [2^k], s_j \xleftarrow{\$} \mathcal{R}, \\ \text{crs}, \kappa_i^{\text{reuse}}, (\kappa_i^{\text{fresh}})^j \\ \text{for } j \in \{1, \dots, t\} \text{ and } i \in I \end{array} \middle| \begin{array}{l} (\text{crs}, \text{st}) \leftarrow \text{Setup}(1^\lambda, 2^k) \\ (\kappa_i^{\text{reuse}} || (\kappa_i^{\text{fresh}})^j) \leftarrow \text{Gen}(\text{crs}, \text{st}, \alpha_j, \beta_j) \\ \text{for } j \in \{1, \dots, t\} \text{ and } i \in \{1, \dots, N\} \end{array} \right\}$$

Next, we provide a construction that is almost identical to that of [ARS24]. The only difference is a careful analysis of each component of the keys and messages in the underlying interactive protocol.

Construction 8 (*N*-Party DPF in the CRS model) *Suppose*

$$\text{sHSS} = (\text{HSS.Setup}, \text{HSS.Enc}, \text{HSS.Hash}, \text{HSS.Eval}, \text{HSS.HEval})$$

is a succinct HSS with reusable public key, as shown in Construction 7. Build an *N*-party DPF over $\mathbb{Z}_2$ as follows:

1. Recap that the dealer runs a multi-round, interactive protocol in its head. In this protocol, each pair of the *N* parties runs a succinct HSS scheme to shift one party's 2^k -size string by another party's k -size input α_i , where α_i for $i = 1, \dots, N$ is the XOR sharing of $\text{Bin}(\alpha)$. Thus each party participates in $(N - 1)$ instances of succinct HSS. The following steps occur entirely within the dealer's head.
2. Run $\text{HSS.Setup}(1^\lambda, 1^{2^k})$ for all the *N* parties, and get the reusable public keys and trapdoors, $(\text{pk}'_i, \text{td}_i)$ for $i = 1, \dots, N$. Append $\text{pk}'_1, \dots, \text{pk}'_N$ to the crs of MPDPF. Append td_i to the reusable key κ_i^{reuse} of party-*i*. Note that the setup algorithm of our succinct HSS is encryptor-specific, thus HSS.Setup is run for *N* times, not $(N + 1)N/2$.
3. Run the HSS.KeyGen algorithm in Construction 7, for each of the *N* parties, and get relatively

$$\begin{aligned} \text{ek}^{(i)} &= (\text{ct}_1^{(i)}, \dots, \text{ct}_{\ell_s}^{(i)}, \text{prfk}_2^{(i)}), \\ \text{sk}^{(i)} &= (s^{(i)}, \text{td}_i, (\text{st}_R^1)^{(i)}, \dots, (\text{st}_R^{\ell_s})^{(i)}, \text{prfk}_1^{(i)}, \text{prfk}_2^{(i)}). \end{aligned}$$

Note that we only need succinct HSS for special RMS programs, thus, the aHMAC is omitted. Append $\text{ct}_1^{(i)}, \dots, \text{ct}_{\ell_s}^{(i)}$ and $\text{prfk}_2^{(i)}$ to the crs of MPDPF.

- Append $s^{(i)}, (\text{st}_R^1)^{(i)}, \dots, (\text{st}_R^{\ell_s})^{(i)}, \text{prfk}_1^{(i)}$ to the reusable key κ_i^{reuse} of party- i . For $i = 1, \dots, N$.
4. Run the HSS.Enc algorithm in Construction 7 on the input α_i of party- i , for each $i \in \{1, \dots, N\}$. The output is $(\mathbf{c}_0^{(i)}, c_1^{(i)}, \dots, c_k^{(i)})$. Append $\mathbf{c}_0^{(i)}$ to the crs of MPDPF. Append $c_1^{(i)}, \dots, c_k^{(i)}$ to each κ_j^{fresh} , for $j = 1, \dots, N$.
 5. The protocol in the dealer's mind has $N - 1$ rounds. In round t , party- $(t + 1)$ serves as the encryptor and parties $1, \dots, t$ each serve as the hasher. When the dealer's mental execution reaches this round, run HSS.Hash for parties $1, \dots, t$ on their respective inputs for this round. The output is $(\text{hv}_t^{(i)}, \psi_t^{(i)})$ for party- i . Append $\text{hv}_t^{(1)}, \dots, \text{hv}_t^{(t)}$ to party- $(t + 1)$'s fresh key $\kappa_{t+1}^{\text{fresh}}$. Append $\psi_t^{(i)}$ to party- (i) 's fresh key κ_i^{fresh} .

Lemma 6 (N -Party DPF in the CRS model). *Assume there exists a succinct HSS scheme (with reusable public key) for special RMS programs, then there exists an N -party DPF scheme in the crs model on domain $[2^k]$ against static, semi-honest adversaries corrupting up to $(N - 1)$ parties. Further, assume the HSS scheme has pk size T_1 , ek size T_2 , sk size T_3 , hv size T_4 , then the DPF scheme has a structured CRS of size $O(N)(T_1 + T_2 + \text{poly}(\lambda))$. For each party, the reusable key size is $O(N)T_3$ and the fresh key size is $O(N)(T_4 + k)$.*

Remark 5. The security proof of MPDPF from succinct HSS is rather involved. In [ARS24], they proved security by arguing that their protocol UC-implements the ideal functionality of MPDPF. In our construction of MPDPF in the CRS model, our succinct HSS scheme integrates into their MPDPF framework in a modular, plug-and-play manner. As long as the underlying HSS satisfies correctness, encryptor privacy, hasher privacy, and pseudorandomness of output shares (which is easy to satisfy for any HSS scheme by appending a PRF key to the public key), the construction is secure.

Combining the above lemma with our succinct HSS with reusable public key for special RMS programs in Corollary 2, we get the following theorem.

Theorem 11 (N -party DPF from bilinear power DDH). *Assuming bilinear power-DDH, there exists an N -party DPF in the CRS model on domain $[2^k]$ against static, semi-honest adversaries corrupting up to $N - 1$ parties, with structured CRS size $O(N(2^k \lambda + \text{poly}(\lambda)))$, reusable key size $O(N \text{poly}(\lambda))$ and fresh key size $O(N(\lambda + k))$.*