

SIMD HSS and aHMAC from Interval Encoding with Application to One-Bit-Per-Gate Garbling

Jaehyung Kim¹, Hanjun Li², Huijia Lin³, Zeyu Liu⁴

¹ Stanford University

² Carnegie Mellon University

³ University of Washington, Seattle

⁴ Yale University

Abstract. Primitives enabling homomorphic computation over *secret-shared* values—Homomorphic Secret Sharing (HSS) and algebraic Homomorphic MACs (aHMAC)—have recently emerged as efficient alternatives to ciphertext-based primitives such as fully homomorphic encryption (FHE) and attribute-based encryption (ABE). Leveraging the distributed nature of secret sharing, direct constructions of HSS and aHMAC are simple, lightweight, avoid costly bootstrapping, and have many applications including one-bit-per-gate garbled circuits.

Despite encouraging progress, all existing direct schemes still lack one key feature: efficient *Single Instruction Multiple Data (SIMD) evaluation*, a capability that has been critical to the efficiency of FHE. This gap leaves the potential of substantial efficiency improvements untapped.

We present the first SIMD evaluation techniques for HSS and aHMAC, based on variants of the RLWE assumption. Using a new *interval coefficient encoding*, our approach embeds $\sqrt{n}$ integer-valued slots per ring element and supports $\sqrt{n}$ -fold batch addition and multiplication in just $O(\log n)$ ring operations, achieving a multiplicative $\tilde{O}(\sqrt{n})$ improvement in amortized efficiency over prior direct constructions. Building on top of these improvements, we show a streamlined one-bit-per-gate SIMD garbling scheme with similar efficiency gains in the online phase.

Our efficiency gains are concrete. Concrete operation counts and microbenchmark based estimates show $6\times$ – $10\times$ improvements in amortized multiplication cost over prior non-SIMD constructions, with up to $25\times$ – $50\times$ speedups for aggregation-heavy workloads such as matrix–vector multiplication. These results demonstrate the practical potential of SIMD techniques for secret-sharing-based homomorphic computation.

Table of Contents

1	Introduction	3
1.1	Our Results in More Detail	4
1.2	Related Works	7
2	Technical Overview	9
2.1	The Challenge of SIMD HSS/aHMAC	9
2.2	Our New Interval Encoding for HSS and aHMAC	10
2.3	Realization of Our Interval Encoding	11
2.4	SIMD One-Bit-Per-Gate Garbled Circuits	12
3	Preliminaries	14
3.1	SIMD Computation	14
3.2	Hard Problems	14
3.3	Fully Homomorphic Encryption	16
4	SIMD for Authenticated Shares from RLWE	17
4.1	Interval Embeddings into $\mathcal{R}$	17
4.2	Authenticated Share Conversion over $\mathcal{R}$	18
4.3	Our RLWE Assumption	23
4.4	Concrete Efficiency of SIMD Operations	23
5	SIMD HSS and aHMAC	25
5.1	HSS Definitions	25
5.2	SIMD HSS Multiplication	27
5.3	SIMD HSS for Bounded Integer RMS	29
5.4	aHMAC Definitions	32
5.5	SIMD aHMAC Multiplication	34
5.6	SIMD aHMAC for Bounded Integer Arithmetic Circuits	36
5.7	Concrete Efficiency	39
6	SIMD Garbling with Amortized 1-Bit per Gate	40
6.1	Garbling Definitions	40
6.2	The Overall Garbling Protocol	41
6.3	The Gate Evaluation Sub-protocol	42
6.4	Security Proof	45
6.5	Concrete Efficiency Analysis	48
7	Evaluation	49
7.1	Comparisons with non-SIMD HSS, aHMAC, and Garbling	49
7.2	Comparisons with SIMD HSS	51
7.3	Additional Comparisons and Discussion	52
A	SIMD Techniques Continued	60
A.1	Non-SIMD Operations	60
A.2	Efficiency Analysis	63
A.3	Additional Discussions	63

1 Introduction

Homomorphic secret sharing (HSS) [BGI16] can be viewed as a secret-sharing analogue of homomorphic encryption (HE) and provides an alternative approach to communication-efficient privacy-preserving two-party computation. At a high level, an HSS scheme enables two parties to locally evaluate functions on secret-shared inputs while keeping the evaluated shares compact, hence enabling efficient output reconstruction. A related primitive, algebraic homomorphic MACs (aHMAC) [ILL25a], can be seen as a secret-sharing analogue of attribute-based encryption (ABE) [SW05] and offers a method for computation over authenticated data. Here, an evaluator locally computes on the plaintext data and their Message Authentication Codes (MACs), while the authenticator locally updates the corresponding MAC keys, again maintaining compact evaluated output MAC and MAC keys, hence supporting efficient verification.

Beyond their conceptual relation to HE and ABE, HSS and aHMAC admit *direct constructions* that exploit the distributed nature of secret sharing. These constructions are often simpler, concretely more efficient [BGI16, BKS19, ACK23, ILL25b], and can be based on diverse assumptions such as DDH and DCR [BGI16, FGJS17, OSY21, ILL25a]. A notable feature is that their parameters remain independent of computation depth, in contrast to HE and ABE constructions that require either parameter growth proportional to the depth or costly bootstrapping [Gen09, HLL23]. As a result, HSS and aHMAC have found applications where the distributed setup arises naturally, including private outsourcing of computation to multiple servers, multi-server PIR [GI14, BGI15, WYG⁺17], sublinear-communication multiparty computation [BGI16, Cou19, CM21], generating correlated pseudorandomness [BCGI18, BCG⁺19], and constraint PRFs [ILL25a].

Together, they offer both privacy and integrity guarantees and yield surprising applications. Notably, a recent line of works [MORS24, ILL25a, ILL25b, MORS25] combined HSS and aHMAC to construct succinct Boolean garbled circuits containing only *one bit per gate*, instead of $\Omega(\lambda)$ bits as in Yao’s garbled circuits [Yao82], and orders of magnitude more efficient than *fully succinct* garbled circuits from FHE plus ABE [GKP⁺13, BGG⁺14, HLL23] or iO [KLW15, BCG⁺18].

Despite this progress, existing direct constructions of HSS and aHMAC **lack efficient support for SIMD-style evaluation**, which has become a central source of efficiency in FHE-based computation. In FHE (e.g., [CKKS17, BGV14, Bra12, FV12, SV14]), SIMD efficiency is achieved by packing many plaintext values into a single encrypted ring element and evaluating them simultaneously via a single homomorphic operation. However, SIMD techniques in FHE rely on NTT or canonical embeddings in the plaintext space $\mathcal{R}_q = \mathbb{Z}_q[x]/\phi(x)$. In contrast, existing direct constructions of HSS, aHMAC, and one-bit-per-gate garbling only support messages in $\mathcal{R} = \mathbb{Z}[x]/\phi(x)$, i.e., coefficient-bounded ring elements. This mismatch prevents a natural transfer of FHE-style SIMD methods.

Indeed, prior HSS work [BKS19] attempts such a transfer by embedding computations in $\mathcal{R}_q$ into the integer ring $\mathcal{R}$, which leads to exponentially growing coefficient bounds—on the order of q^d for degree- d computations—making the method practical only for low-degree computations and unable to support SIMD evaluation at unbounded depth.¹ This leaves open the central question:

Can we obtain efficient constructions of HSS and aHMAC that support SIMD evaluation at unbounded depth, without relying on SIMD FHE?

¹ Unbounded-depth SIMD evaluation requires parameters that remain independent of the computation depth.

	NTT/iNTT ($\mathcal{R}_q$)	Elm-Mult ($\mathcal{R}_q$)	Elm-Add ($\mathcal{R}$)	Rounding	Automorphism ($\mathcal{R}$ or $\mathcal{R}'$)
Non-SIMD Homomorphic Evaluation per Mult or Gate					
HSS mult [BKS19]	4	4	4	2	0
aHMAC mult [ILL25a]	6	7	4	1	0
Bit-GC, online [ILL25b, LWYY25]	26	27	24	11	0
$\sqrt{n}$-fold SIMD Homomorphic Evaluation per Mult or Gate					
HSS $\sqrt{n}$ -mult Construction 7	$7 + \frac{3\log(n)}{2}$	$6 + \log(n)$	$6 + \log(n)$	$3 + \frac{\log(n)}{2}$	$1 + \frac{\log(n)}{2}$
aHMAC $\sqrt{n}$ -mult Construction 10	$12 + \frac{3\log(n)}{2}$	$11 + \log(n)$	$8 + \log(n)$	$3 + \frac{\log(n)}{2}$	$2 + \frac{\log(n)}{2}$
Bit-GC, online, $\sqrt{n}$ -gate Figure 5	$41 + 12\log(n)$	$37 + 8\log(n)$	$34 + 8\log(n)$	$16 + 4\log(n)$	$7 + 2\log(n)$
$\sqrt{n}$-fold SIMD Homomorphic Evaluation per Dimension-w Inner Product					
HSS $\sqrt{n}$ -mult Section 5.7	$4w + \frac{3\log(n)}{2} + 3$	$4w + \log(n) + 2$	$4w + \log(n) + 2$	$2w + \frac{\log(n)}{2} + 1$	$\frac{\log(n)}{2} + 1$
aHMAC $\sqrt{n}$ -mult Section 5.7	$6w + \frac{3\log(n)}{2} + 3$	$7w + \log(n) + 4$	$4w + \log(n) + 4$	$w + \log(n) + 3$	$\frac{\log(n)}{2} + 2$

Table 1: We list concrete number of different ring operations needed for homomorphic evaluation per multiplication/gate of the state-of-the-art non-SIMD schemes and our SIMD schemes. We also show the optimized number of operations for SIMD inner product of our HSS and aHMAC. Each scheme has two algorithms (two parties in HSS, key and tag evaluation algorithms in aHMAC, garbling and evaluation algorithms in garbling schemes). The numbers listed for each primitive are the maximum among the two algorithms. $\mathcal{R}_q = \mathbb{Z}_q[X]/X^n + 1$ is the ring associated with RLWE, while $\mathcal{R}'$ is a related ring $\mathcal{R}' = \mathbb{Z}[X]/X^{n'} - 1$ for $n' > n$.

Our contributions. We present the first direct constructions of SIMD HSS and SIMD aHMAC that do not rely on FHE, together with corresponding SIMD acceleration for one-bit-per-gate garbling. Our approach is based on a new packing method, which we call *interval coefficient encoding*, that enables $\sqrt{n}$ -way SIMD evaluation while operating entirely within the message space $\mathcal{R}$. Our homomorphic batch addition and multiplication then achieve a multiplicative factor of $\tilde{O}(\sqrt{n})$ improvement in amortized efficiency over prior state-of-the-art lattice-based HSS and aHMAC schemes [BKS19, ILL25a], and yield improvements of the same order in the *online phase* of one-bit-per-gate Boolean garbling and evaluation over [ILL25b, LWYY25]. Concretely, we demonstrate a $\sim 6\times$ improvement over plain HSS and $\sim 10\times$ improvements over plain aHMAC and one-bit-per-gate garbling in Section 7 and Table 2.

The new interval encoding method relies on circular RLWE with entropic secrets satisfying mild requirements (formally described in Figure 1). We discuss the assumptions in more detail later in the introduction.

1.1 Our Results in More Detail

We briefly summarize the main guarantees achieved by our SIMD constructions. All of our schemes support arithmetic over bounded integers in a range $[B]$, composed of addition and multiplication operations. Such bounded integer computation suffices to realize arbitrary Boolean computation with $B = 2$ and only constant overhead, and therefore captures the primary use cases of HSS, aHMAC, and one-bit-per-gate garbling.

SIMD HSS. Prior direct HSS constructions without FHE (e.g., [BGI16, ADOS22, BKS19]) support only Restricted Multiplication Straightline (RMS) programs, where each multiplication

involves at least one input element. Building on the RLWE-based construction of [BKS19], we obtain the first construction that additionally enables efficient $\sqrt{n}$ -fold SIMD evaluation while maintaining parameters independent of computation depth.

Theorem 1 (Informal). *Assuming circular RLWE with entropic secrets over a ring $\mathcal{R}_q$ of dimension n , the scheme in Section 5 is a (direct) HSS scheme supporting $\sqrt{n}$ -fold SIMD evaluation of restricted multiplication programs, where batched addition requires a single local ring addition, and batched multiplication requires $O(\log n)$ local ring operations, summarized in Table 1.*

This yields an amortized $\tilde{O}(\sqrt{n})$ improvement per arithmetic operation over prior direct HSS schemes [BKS19]. The primary tradeoff is that our construction relies on circular RLWE with entropic secrets satisfying mild additional requirements, rather than plain (circular) RLWE. Our SIMD extensions for aHMAC and one-bit-per-gate garbling below require analogous strengthenings of the underlying assumptions used in the corresponding prior works. We discuss these assumption refinements and their standard mitigations in more detail later in the introduction.

SIMD aHMAC. The algebraic homomorphic MAC (aHMAC) [ILL25a] supports two complementary forms of homomorphism. First, *tag evaluation* transforms input tags $\sigma_{\mathbf{x}}$ into an algebraic output tag of the form $\sigma_f = \mathbf{s} \cdot y + \mathbf{k}_f$ for $y = f(\mathbf{x})$, while using the input $\mathbf{x}$ in the clear. Second, *key evaluation* derives the corresponding MAC key $\mathbf{k}_f$ directly from the secret MAC key, without access to $\mathbf{x}$. We extend the RLWE-based construction of [ILL25a] to support the same $\sqrt{n}$ -fold SIMD evaluation.

Theorem 2 (SIMD, aHMAC, Informal). *Assuming circular power RLWE with entropic secrets over a ring of dimension n , the scheme in Section 5 is an algebraic homomorphic MAC scheme supporting $\sqrt{n}$ -fold SIMD evaluation for both homomorphic tag and key evaluation. In both homomorphic procedures, batched addition requires one ring addition and batched multiplication requires $O(\log n)$ ring operations.*

As summarized in Table 1, this gives an amortized $\tilde{O}(\sqrt{n})$ reduction in per-operation complexity compared to prior work [ILL25a].

SIMD One-Bit-Per-Gate Garbled Circuits. Our SIMD HSS and SIMD aHMAC constructions together yield a corresponding acceleration for one-bit-per-gate Boolean garbling in the standard offline/online model. This online–offline paradigm is common across cryptography, appearing in secure multiparty computation, garbling schemes, fully homomorphic encryption, and SNARKs, where a relatively expensive preprocessing phase is acceptable provided that the latency-critical online phase remains lightweight. In our garbling scheme, an offline phase (independent of both the circuit C and the input x) expands PRG seeds into one pseudorandom bit per wire using FHE, while the online phase evaluates only a constant number of HSS and aHMAC operations per gate, which is efficient. We improve the online efficiency, through *batched* garbling and evaluation enabled by $\sqrt{n}$ -fold SIMD HSS and aHMAC.

Theorem 3 (Informal). *Assume circular power RLWE with entropic secrets as in the HSS and aHMAC constructions above. There exists a one-bit-per-gate garbling scheme for Boolean circuits in the offline/online model that supports $\sqrt{n}$ -fold SIMD garbling and evaluation in the online phase, with*

- garbled circuit size $\sqrt{n} \cdot |C| + \text{poly}(n)$, and
- online garbling and evaluation of each $\sqrt{n}$ -fold Boolean gates using $O(\log n)$ ring operations.

Consequently, the amortized online cost per Boolean gate improves by a factor of $\tilde{O}(\sqrt{n})$ over prior RLWE-based one-bit-per-gate garbling schemes [LWYY25, ILL25b]. Table 1 shows concrete operation counts for a more computationally optimized version that sends 2 bits per gate described in Section 6.5.

Our focus is on improving online efficiency. For the offline phase, we modularize the required interface and provide a compatible instantiation using general FHE, without attempting further optimization. A recent independent work [ZLY⁺26] gives a concretely efficient SIMD-FHE-based realization of the offline phase, but does not accelerate the online phase and therefore achieves online efficiency comparable to prior RLWE-based constructions [LWYY25, ILL25b]. Their offline optimizations compose directly with our SIMD online algorithms.

SIMD Inner Products. Beyond gate-by-gate SIMD evaluation, our HSS and aHMAC schemes are particularly efficient for workloads in which many products are aggregated by sums². Such patterns, including matrix–vector multiplication, convolutions, and inner products, are central to applications such as privacy-preserving machine learning [BCH⁺24, jLHH⁺21, HS18, VZ25, BN25, BCH⁺25, LKL⁺22, LLKN23]. In these settings, most of the cost of each SIMD multiplication can be deferred until after aggregation, yielding an additional factor- w , overall factor- $\tilde{O}(\sqrt{n} \cdot w)$, improvement (see Table 1 and Figure 7), where w denotes the number of aggregated products.

Non-SIMD Computations. We also develop homomorphic *rotation*, *selection*, and *packing/unpacking* procedures over the $\sqrt{n}$ SIMD slots. These operations, combined with standard techniques such as in [GHS12a], extend our framework to support general non-SIMD computations.

Performance Evaluation via Microbenchmarking. A central feature of our work is that the efficiency gains are concrete. In Section 7, we estimate the amortized runtime per multiplication under a workload of 2^{20} multiplications, targeting 40-bit statistical security and 128-bit computational security. Our methodology benchmarks the runtime of basic ring operations on a CPU implementation and extrapolates to multiplication in each scheme using the operation counts reported in Table 2. This yields estimated amortized costs of approximately $60\mu s$ per multiplication for SIMD HSS, $138\mu s$ for SIMD aHMAC, and $381\mu s$ for the online phase of SIMD garbling. These correspond to empirical improvements of roughly $6\times$ to $10\times$ over the corresponding non-SIMD constructions in [BKS19, ILL25a, ILL25b]. Compared with the prior SIMD HSS approach for low-degree polynomials in [BKS19], our scheme achieves more than a $100\times$ advantage when evaluating a restricted-multiplication straightline (RMS) program implementing a shallow Boolean circuit of depth $D = 15$ (see Figure 6). For workloads that aggregate many products by summation, such as matrix–vector multiplication and inner products, our HSS and aHMAC constructions become even more efficient, achieving speedups of up to $25\times$ for HSS and $50\times$ for aHMAC. Table 2 also shows amortized runtime on GPU and dedicated hardware estimated using existing micro-benchmarks in the literature. These results highlight that SIMD interval encoding can substantially improve the practical performance of HSS and aHMAC in real-world applications.

Our Interval Encoding. At the heart of our construction is a new packing method, *interval encoding*, which directly embeds $\sqrt{n}$ integer plaintext slots into the coefficients of a ring element in $\mathcal{R}$. Unlike the canonical or NTT embeddings used in FHE, ring multiplication translates to *convolution* on coefficients, which at first glance seems ill-suited for SIMD computation.

² Note that this additional improvement is achieved over bounded integer computations, which is not applicable to SIMD Boolean garbling.

To address the gap between SIMD and convolution, we design three encodings, Types A, B, and C, that place $\sqrt{n}$ -dimensional vectors into different subsets of coefficients. Multiplying a Type A and Type B polynomial yields a polynomial with tensor product of the input vectors as the coefficients. We then apply a *homomorphic clear-coefficient* procedure to filter out irrelevant terms, retaining only the coordinate-wise products encoded in Type C. Finally, a *homomorphic type-conversion* step maps Type C encodings back into Type A (or B), thereby enabling iterative SIMD computation.

To implement these procedures efficiently, we introduce new techniques that leverage ring automorphisms, trace maps, and a novel use of *ring switching*. While automorphisms and trace have appeared in the FHE literature, they have not been used to support SIMD, nor in secret-sharing based primitives, and ring switching is entirely new. The latter exploits the distributed nature of secret sharing to temporarily switch to rings $\mathcal{R}'$, that are different from the RLWE ring $\mathcal{R}$, which provides additional algebraic structure that make the overall construction possible.

Discussion on assumptions. Prior efficient lattice-based constructions of HSS, aHMAC, and one-bit-per-gate garbling already rely on two refinements of the standard RLWE assumption:

- *Circular security.* RLWE samples are used in a circular manner to hide functions of the secret. Circular security is standard in the literature, particularly in FHE constructions.
- *Power variant.* A power variant of RLWE—where the adversary observes samples of the form $(a, sa + e, s^2a + e)$ (Definition 2)—is required for existing aHMAC and garbling schemes. This assumption was introduced in [ILL25b], inspired by the power-DDH assumption.

To enable SIMD evaluation via interval coefficient encoding, our work requires an additional refinement:

- *Entropic secrets*, satisfying that a fixed subset of coefficients is set to zero (while the remaining coefficients have no constraints except for being small relative to the modulus); see Requirement 1.

Entropic RLWE has been studied previously, with reductions to standard RLWE known for the *search* problem but not for the *decision* problem (see [BD20]). Our specific entropic distribution is closely related to structured secret distributions widely used in the FHE literature (e.g., [GHPS12, BCK⁺23, LW23, CKP25, LW24]), as discussed in Section 3.2.

Combining these refinements yields concretely efficient SIMD constructions of HSS, aHMAC, and one-bit-per-gate garbling. The asymptotic efficiency gain, however, can still be based on weaker assumptions, namely entropic RLWE, via standard transformations. In particular, circular security can be removed in leveled variants of SIMD HSS, SIMD aHMAC, and SIMD garbling using known techniques [MORS25, ILL25b, ILL25a], at the cost of increasing the size of the public parameters (ie., evaluation keys) in aHMAC, shares in aHMAC, and garbled circuits, by a depth-dependent term, and increased computation in aHMAC and garbling. The power variant can be circumvented using the transformation introduced in [MORS25] and later applied in [ILL25a], while roughly doubling the computation costs. The resulting constructions after these two transformations still achieve the $\tilde{O}(\sqrt{n})$ -factor improvement on amortized cost.

We present our schemes under the stronger assumptions, for simplicity of exposition and consistency with our concrete efficiency evaluation.

1.2 Related Works

In the introduction, we compared our SIMD schemes with prior direct RLWE-based HSS, aHMAC, and succinct garbling schemes, as they provide the most meaningful baseline. Schemes

based on other assumptions, such as DDH, DCR, or plain LWE (e.g., [BGI16, BGI17, ADOS22, MORS24, MORS25, ILL25a, ILL25b]), are far less efficient in practice since their underlying operations (e.g., group exponentiation, matrix–matrix multiplication) are significantly more expensive than ring operations.

Comparison with SIMD FHE. It is well known that HSS can be constructed from special FHE [BGI15, BGI⁺18, DHRW16]. Compared to SIMD FHE, which can achieve full n -fold packing, our scheme supports only $\sqrt{n}$ -fold packing. However, this is not a major limitation in practice: for common parameters $n = 2^{13}$ or 2^{14} , our schemes achieve SIMD widths of 64–128, which is a natural scale for applications such as (single or multi-server) batch-PIR, where a user simultaneously fetches many entries from a database. Indeed, FHE with sparse (rather than full) packing has been considered, but comes with much worse amortized efficiency [CHK⁺18, CH18].

From an efficiency standpoint, a fully fair comparison is difficult, but the following gives a useful baseline: without SIMD, TFHE costs ~ 10 ms per Boolean gate [CGGI20]; with medium SIMD ($\leq 2^{11}$), amortized TFHE bootstrapping achieves ~ 4 ms per gate [GP25]; and with very large SIMD ($\geq 2^{15}$), the CKKS-based bootstrapping achieves $\sim 20\mu$ s per gate [BCKS24]. In contrast, our SIMD HSS/aHMAC achieves tens of microseconds per multiplication, amortized over only 2^6 inputs, and for structured computations such as matrix–vector multiplication the cost drops to $\sim 13\mu$ s per multiplication (see Table 2 and Fig. 7). Thus, for applications where massive SIMD is unnecessary or low latency is preferred (e.g., [LT22, LTW24, Zam22]), our approach offers better amortized efficiency.

Last but not least, our approach offers other important advantages: simplicity, low latency (no bootstrapping), and integrity guarantees.

Comparison with Garbling Schemes. Compared to Yao’s garbled circuits and its optimizations, our scheme is more succinct (one bit per gate vs. $\Omega(\lambda)$ bits), but per-gate evaluation is slower. Yao’s scheme, relying only on symmetric-key primitives, can garble/evaluate each gate in ~ 100 ns using AES-NI [BHKR13], while our estimates for the online phase, amortized, are $\sim 400\mu$ s on CPU, $\sim 30\mu$ s on GPU, and ~ 400 ns on dedicated hardware. These numbers nevertheless indicate clear potential for practical deployments where communication, not local computation, is the main bottleneck (e.g., blockchain).

Fully succinct garbled circuits achieve theoretically optimal size, but because they rely on heavy machinery such as FHE+ABE or iO, their concrete costs remain prohibitively large. In contrast, our scheme strikes a favorable balance, combining succinctness with concrete efficiency.

Recent work on one-bit-per-gate garbling A very recent work also improves one-bit-per-gate garbling [ZLY⁺26]. However, its primary focus lies in the offline phase. In particular, it proposes concretely efficient methods for instantiating the offline phase of such garbling using SIMD-capable FHE, while the online phase remains non-SIMD, as in prior works [ILL25b, LWYY25, MORS25]. Additionally, it introduces techniques enabling free-XOR for this class of low-communication garbling, as well as an alternative construction that achieves improved runtime at the cost of increasing communication to five bits per gate instead of one. These techniques are complementary to our results, as our work focuses on enabling SIMD in the online phase, and we expect that our techniques could similarly accelerate their online phase, yielding an amortized efficiency improvement of approximately $\sim 10\times$ per gate, as shown in Table 2. In Section 7.3, we further discuss (1) how their offline-phase instantiation can be integrated into our construction to support SIMD evaluation online, and (2) how our SIMD techniques for the online phase could potentially accelerate their offline-phase instantiation as well.

2 Technical Overview

Our starting points are the existing RLWE-based constructions of a homomorphic secret sharing (HSS) scheme from [BKS19] and an algebraic homomorphic MAC (aHMAC) scheme from [ILL25b]. We observe that while both schemes focus on realizing arithmetic computations, i.e. additions and multiplications, over bounded integers or over $\mathbb{Z}_2$, their underlying constructions actually natively support computations over *polynomial ring elements with bounded coefficients*. The polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ is the ring $\mathcal{R}_q$ associated with the underlying RLWE without modulo q , where n is a power-of-two; only the constant coefficient is utilized for computation in those constructions. Our goal is to efficiently embed multiple instances of computations over bounded integers into a single instance of computation over $\mathcal{R}$.

2.1 The Challenge of SIMD HSS/aHMAC

Using computation over $\mathcal{R}$ to realize SIMD computations over bounded integers has been studied extensively in the fully homomorphic encryption (FHE) literature. However, as we explain below, the techniques developed in the context of FHE cannot be efficiently applied in the context of HSS or aHMAC. We therefore develop new embedding and evaluation techniques in this work, as explained in the next subsections. We now review the HSS and aHMAC constructions in more detail.

In the setting of HSS, two parties P_0, P_1 are initially both provided with some ciphertexts $\text{ct}_s(x_i)$ of the inputs $\{x_i \in \mathcal{R}\}$ under a (global) RLWE secret s . Additionally, they are respectively provided with additive shares of the secret $\langle s \rangle_0, \langle s \rangle_1$ over $\mathcal{R}$, representing an authenticated share $\langle \mathbf{s} \cdot 1 \rangle$ of an initial value 1 with respect to the global secret $\mathbf{s} := (1, s)$. Two types of computations are supported:

- Addition between two authenticated shares $\langle \mathbf{s} \cdot y \rangle, \langle \mathbf{s} \cdot y' \rangle$ (jointly held by the parties) gives $\langle \mathbf{s} \cdot (y + y') \rangle$.
- Multiplication between an authenticated share $\langle \mathbf{s} \cdot y \rangle$, and an input ciphertext $\text{ct}_s(x_i)$ gives another share $\langle \mathbf{s} \cdot x_i \cdot y \rangle$ of the product $x_i \cdot y$.

Starting from the initial authenticated share $\langle \mathbf{s} \cdot 1 \rangle$ of 1, the parties can locally evaluate any restricted multiplication straightline (RMS) program P , consisting of a sequence of the above supported computations, to derive an authenticated share $\langle \mathbf{s} \cdot z \rangle$ of the final result $z = P(\{x_i\})$. To reveal z , the parties reconstruct to each other the first component of the authenticated share, which is $\langle z \rangle$.

The setting of aHMAC similarly involves two parties P_0, P_1 , who are initially both provided with some public data pd_s , which can be viewed as a circular ciphertext of a global secret s , and jointly authenticated shares $\{\langle \mathbf{s} \cdot x_i \rangle\}$ with respect to the global secret $\mathbf{s} := (1, s)$. Additionally, P_1 is provided with the inputs $\{x_i \in \mathcal{R}\}$ in clear, and P_0 is provided with the global secret s . Two types of computations are supported (using only local evaluations by the parties):

- Addition between authenticated shares $\langle \mathbf{s} \cdot y \rangle, \langle \mathbf{s} \cdot y' \rangle$ gives $\langle \mathbf{s} \cdot (y + y') \rangle$.
- Multiplication between authenticated shares $\langle \mathbf{s} \cdot y \rangle, \langle \mathbf{s} \cdot y' \rangle$ gives $\langle \mathbf{s} \cdot y \cdot y' \rangle$.

In the end, the parties can locally evaluate any arithmetic circuit C to derive an authenticated share $\langle \mathbf{s} \cdot z \rangle$ of the final result $z = C(\{x_i\})$. P_1 can then authenticate this result z to P_0 by revealing its share.

We now explain why SIMD techniques from FHE literature fail. They fall into two categories. The first method relies on FHE schemes to implement a message space $\mathcal{R}_p$ for some prime p such that the polynomial ring splits completely [SV14, BV11, Bra12, FV12]: $\mathcal{R}_p \simeq \mathbb{Z}_p \times \dots \times \mathbb{Z}_p$. Homomorphic computations over $\mathcal{R}_p$ can therefore implement n SIMD computations over $\mathbb{Z}_p$. However, adapting this technique in the context of HSS or aHMAC would require efficiently implementing *homomorphic modulo reduction by p on authenticated shares over $\mathcal{R}$* , which is a difficult open problem.

We note that the work of [BKS19] proposes a work-around for HSS by delaying the modulo reduction till the end, when the parties jointly hold authenticated shares $\langle \mathbf{s} \cdot z \rangle$ of the final result z . Since the parties only need $\langle z \rangle$ for reconstruction, they can easily do modulo reduction on $\langle z \rangle$ before reconstruction. The downside is that without modulo reductions on intermediate values y , their coefficients grow exponentially in the computation depth, requiring huge parameter settings and slow computations in the HSS scheme.

The second method relies on FHE schemes with message space $\mathcal{R}$ to approximately implement SIMD computations over complex numbers [CKKS17]: $\mathbb{C} \times \dots \times \mathbb{C}$. This method avoids homomorphic modulo reduction (by p) over the message space, but instead needs homomorphic rounding to prevent exponentially growing coefficients in the intermediate evaluation results. Again, adapting this technique in the context of HSS or aHMAC would require efficiently implementing *homomorphic rounding over authenticated shares over $\mathcal{R}$* , which is basically equivalent to implementing homomorphic modulo reduction.

2.2 Our New Interval Encoding for HSS and aHMAC

Instead of relying on the above techniques from FHE, we design a new packing technique, called *interval encodings*, that embeds $T = O(\sqrt{n})$ bounded integers directly in the coefficients of an element in $\mathcal{R}$ in some special structures. We give more details on this encoding below, and then present homomorphic evaluation procedures over such encodings.

Interval Encodings. To motivate our interval encoding, first consider two arbitrary subsets of coefficient positions, $A = \{a_0, \dots, a_{T-1}\}$, $B = \{b_0, \dots, b_{T-1}\} \subseteq [n]$, each of size T , and embed two integer vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^T$ respectively into the coefficients of two ring elements $\mathcal{E}_{\mathbf{x}}^A, \mathcal{E}_{\mathbf{y}}^B \in \mathcal{R}$:

$$\mathcal{E}_{\mathbf{x}}^A := \sum_{i \in [T]} \mathbf{x}[i] \cdot X^{a_i=i} \text{ and } \mathcal{E}_{\mathbf{y}}^B := \sum_{i \in [T]} \mathbf{y}[i] \cdot X^{b_i=(S-1) \cdot i}.$$

We observe the coefficients of the product $\mathcal{E}_{\mathbf{x}}^A \cdot \mathcal{E}_{\mathbf{y}}^B$ contain all of the desired SIMD multiplication results $\mathbf{x}[i] \cdot \mathbf{y}[i]$ at positions $a_i + b_i$, as long as there are no overlaps between $a_i + b_j$ for $i, j \in [T]$. Choosing $A := \{0, 1, \dots, T-1\}$, and $B = \{0, (S-1), \dots, (T-1) \cdot (S-1)\}$, for some $\sqrt{n} \geq S \geq T$ satisfy this requirement. We call the embedding of integer vectors into coefficient positions in A, B respectively Type A and Type B encodings.

While the product between a Type A and Type B encoding contains all necessary information of the desired SIMD (i.e. element-wise) product $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$, it is no longer a Type A or Type B encoding and cannot be used for further SIMD evaluations. In the following, we describe two homomorphic procedures (on authenticated shares) that converts such a product back into a Type A encoding (which can then be converted to Type B as described below):

- **Trace:** removes (i.e., sets to 0) coefficients other than the positions in $C := \{c_i = a_i + b_i = S \cdot i\}_{i \in [T]}$. We call such an encoding, with only non-zero coefficients at positions in C a Type

C encoding.

$$\mathcal{E}_{\mathbf{x} \odot \mathbf{y}}^C = \text{Trace}(\mathcal{E}_{\mathbf{x}}^A \cdot \mathcal{E}_{\mathbf{y}}^B), \text{ where } \mathcal{E}_{\mathbf{z}}^C := \sum_{i \in [T]} \mathbf{z}[i] \cdot X^{c_i=S \cdot i}.$$

- **ConvCA**: converts a Type C encoding back to Type A.

Using interval encoding for HSS and aHMAC. Implementing the two homomorphic operations above over authenticated shares suffice for SIMD HSS over bounded integers. In particular, the two parties initially will both hold ciphertexts of Type B encodings of SIMD inputs $\text{ct}_s(\mathcal{E}_{\mathbf{x}}^B)$, and jointly an authenticated share of 1, in Type A encoding $\langle \mathbf{s} \cdot \mathcal{E}_1^A \rangle$. Two types of SIMD computations are supported:

- Addition between two authenticated shares $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}'}^A \rangle$ gives $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}+\mathbf{y}'}^A \rangle$.
- Multiplication (using the original HSS over $\mathcal{R}$) between an authenticated share $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle$, and an input ciphertext $\text{ct}_s(\mathcal{E}_{\mathbf{x}}^B)$, gives $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \cdot \mathcal{E}_{\mathbf{x}}^B \rangle$. Then applying homomorphic **Trace** and **ConvCA** over this authenticated share gives $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y} \odot \mathbf{x}}^A \rangle$, where $\odot$ denotes component-wise (i.e. SIMD) products.

On the other hand, for SIMD aHMAC over bounded integers, we need one more procedure: **ConvAB** from Type A encoding to Type B. This is because in aHMAC, the two parties initially both hold authenticated shares of Type A encoding $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}_i}^A \rangle$ (instead of one Type B ciphertext as for HSS). Thus, we need to first transform one of the shares to Type B. Given **ConvAB** (discussed in more detail below), aHMAC can be achieved, where two types of SIMD computations are supported as follows:

- Addition is the same as SIMD HSS.
- Multiplication between two authenticated shares $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}'}^A \rangle$ involve first converting one of them into Type B: $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^B \rangle$. Then, applying the original aHMAC multiplication over $\mathcal{R}$ gives $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}'}^A \cdot \mathcal{E}_{\mathbf{y}}^B \rangle$. Finally, applying homomorphic **Trace** and **ConvCA** gives $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y} \odot \mathbf{y}'}^A \rangle$.

2.3 Realization of Our Interval Encoding

With these in mind, the remaining step is to show how homomorphic trace and type conversions are realized.

Realizing homomorphic trace. Homomorphic trace evaluation is widely used in the context of fully homomorphic encryption (such as [CDKS19]). It can be implemented efficiently via roughly $\log S$ iterative evaluation of homomorphic automorphisms, assuming S is a power-of-two.

We now describe homomorphic automorphism implemented using the original HSS multiplication over $\mathcal{R}$.³ It converts an authenticated share $\langle \mathbf{s} \cdot v \rangle$ into $\langle \mathbf{s} \cdot \varphi(v) \rangle$ for a given ring automorphism φ .

- Initially prepare a (circular) ciphertext $\text{ct}_s(\varphi^{-1}(s))$.
- Apply the original HSS multiplication over $\mathcal{R}$ between $\langle \mathbf{s} \cdot v \rangle$ and $\text{ct}_s(\varphi^{-1}(s))$ to derive $\langle \mathbf{s} \cdot \varphi^{-1}(s) \cdot v \rangle$, the first component of which is $\langle \varphi^{-1}(s) \cdot v \rangle$.
- Both parties locally apply φ on their shares of $\langle \varphi^{-1}(s) \cdot v \rangle$ to derive $\langle s \cdot \varphi(v) \rangle$. The parties also take their given share $\langle v \rangle$ and apply φ to derive $\langle \varphi(v) \rangle$. Combining the two results gives the desired $\langle \mathbf{s} \cdot \varphi(v) \rangle$.

³ Although the described homomorphic automorphism uses HSS multiplications over $\mathcal{R}$, it's also compatible with aHMAC.

Homomorphic trace can then be realized through $\log S$ such automorphism steps.

Type Conversions. Conversion from Type A to Type B and vice versa are relatively simple, as they can be defined via a single (homomorphic) automorphism. Let $\psi_S : \mathcal{R} \rightarrow \mathcal{R}$ be defined as $X \mapsto X^{(S-1)}$:

$$\psi_S(\mathcal{E}_{\mathbf{x}}^A) = \mathcal{E}_{\mathbf{x}}^B \text{ and } \psi_S^{-1}(\mathcal{E}_{\mathbf{x}}^B) = \mathcal{E}_{\mathbf{x}}^A$$

Applying the above homomorphic automorphism gives the desired conversions.

The Type C to Type A conversion is more involved, because our desired mapping $\psi : X^S \mapsto X$ is *not* an automorphism over $\mathcal{R}$, where S is a power-of-two. Applying the above homomorphic automorphism procedure does not work. To solve this issue, we apply a special type of “ring switching” on the authenticated share of Type C encodings for this conversion.⁴ We choose a different ring $\mathcal{R}' = \mathbb{Z}[X]/(X^{n'} - 1)$ in which the mapping $X^S \mapsto X$ is an automorphism. In more detail:

- Parties start with authenticated shares $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^C \rangle$ over $\mathcal{R}$. It first key switches to $\tilde{\psi}(s)$ where $\tilde{\psi} = X \mapsto X^S : \mathcal{R}' \rightarrow \mathcal{R}'$ is an automorphism over $\mathcal{R}'$. They obtain $\langle \tilde{\psi}(s) \cdot \mathcal{E}_{\mathbf{z}}^C \rangle$
- Assuming the degree of $\tilde{\psi}(s)$ and $\mathcal{E}_{\mathbf{z}}^C$ are both small, in particular their sum is smaller than n , then the share $\langle \tilde{\psi}(s) \cdot \mathcal{E}_{\mathbf{z}}^C \rangle$ can be viewed as over $\mathbb{Z}[X]$ instead of $\mathcal{R}$.
- Parties locally reduce their shares over $\mathcal{R}'$, and apply $\tilde{\psi}^{-1}$ locally to derive shares $\langle \mathbf{s} \cdot \tilde{\psi}^{-1}(\mathcal{E}_{\mathbf{z}}^C) \rangle$ over $\mathcal{R}'$. Note that $\mathcal{E}_{\mathbf{z}}^A = \tilde{\psi}^{-1}(\mathcal{E}_{\mathbf{z}}^C)$ over $\mathcal{R}'$ as desired. Assuming again the degree of $\mathbf{s}$ and $\mathcal{E}_{\mathbf{z}}^A$ over $\mathcal{R}'$ are small, in particular their sum smaller than n' , then the share $\langle \mathbf{s} \cdot \tilde{\psi}^{-1}(\mathcal{E}_{\mathbf{z}}^C) \rangle = \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^A \rangle$ can again be viewed as over $\mathbb{Z}[X]$ instead of $\mathcal{R}'$.
- Parties locally reduce the shares over $\mathcal{R}$ gives $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^A \rangle$ in $\mathcal{R}$.

Note that this ring switching process is possible because the shares are simply shares of the coefficients of the ring elements, which allows the elements to be seamlessly transformed among different rings. Furthermore, crucially, the RLWE assumption we need does *not* rely on this intermediate $\mathcal{R}'$, since the ring is only used for the shares, not the ciphertexts.

To satisfy the low-degree requirements from the above, we can set, for example, $S = T \leq \sqrt{n}/2$ such that $S = T$ are powers-of-two and s to satisfy $\deg(\tilde{\psi}(s)) < n/2 + S$ and $\deg(s) < n' - T$ so that we have

$$\deg(\tilde{\psi}(s)) + \deg(\mathcal{E}_{\mathbf{z}}^C) < n/2 + S + S \cdot (T - 1) \leq n,$$

$$\deg(s) + \deg(\mathcal{E}_{\mathbf{z}}^A) < n' - T + T \leq n'.$$

Note that s can satisfy the requirements by setting at most $n/2$ coefficients to zero. In other words, s can have at least $n/2$ non-zero coefficients. Due to this restriction on the distribution of s , we need a circular secure entropic RLWE assumption with such secret distributions for our SIMD HSS scheme. We need an analogous circular power-RLWE for our SIMD aHMAC and GC schemes.

2.4 SIMD One-Bit-Per-Gate Garbled Circuits.

Lastly, we briefly discuss how our garbled circuit is constructed. Recent works [LWYY25, ILL25b, MORS25] show how to garble Boolean circuits by sending essentially one-bit-per-gate, based on a variety of (circular variants of) assumptions DDH, DCR, NTRU, or RLWE. We present a scheme that draws inspiration from the two lattice-based constructions of [LWYY25, ILL25b].

⁴ Note that the ring switching technique in FHE [GHPS12] greatly *differs* from our scheme.

In more detail, the scheme of [ILL25b] combines HSS and aHMAC, while the scheme of [LWYY25] relies on the Gentry-Sahai-Waters (GSW) FHE [GSW13] with additionally designed distributed decryption algorithms.⁵ Towards better concrete efficiency and conceptual clarity, our scheme splits the garbling and evaluation algorithms into pre-processing (independent of C or x) and online phases.

- The pre-processing phases perform FHE evaluation of a PRG, similarly to [LWYY25], producing ciphertexts of pseudorandom wire masks.
- The online phases perform aHMAC and HSS evaluations, similarly to [ILL25b], using the evaluated FHE ciphertexts of wire masks.

We now give more detail on our garbling scheme with pre-processing. We first briefly recall the framework of [ILL25b] as our starting point. Assuming appropriate setup steps, the garbler and the evaluator respectively take the following steps for each gate $g \in C$ (with input wires i, j and output k):

0. The parties jointly hold authenticated shares $\langle \mathbf{s} \cdot \bar{x}^{(i)} \rangle, \langle \mathbf{s} \cdot \bar{x}^{(j)} \rangle$ w.r.t. a global secret $\mathbf{s} = (1, s)$ of masked inputs $\bar{x}^{(i)}, \bar{x}^{(j)}$, where each wire mask is derived from a PRF or local PRG:

$$\bar{x}^{(i)} \leftarrow x^{(i)} \oplus r^{(i)}, \text{ where } r^{(i)} \leftarrow \text{PRF}(\text{key}, i).$$

The evaluator additionally holds $\bar{x}, \bar{y}$ in the clear.

1. The parties first apply aHMAC evaluations to derive shares of an one-hot vector $\langle \mathbf{s} \cdot \mathbf{z} \rangle$ where $\mathbf{z}[i] = 1$ if and only if the two masked input values $(\bar{x}^{(i)}, \bar{x}^{(j)})$ encode the index i , for $i \in [4]$.
2. The parties then apply HSS evaluations between the authenticated shares $\langle \mathbf{s} \cdot \mathbf{z} \rangle$ and a set of ciphertexts $\{\text{ct}_s(\text{key}[i])\}$ encrypting the PRF key to evaluate the following function $f^{(k)}$

$$f^{(k)}(\text{key}, \mathbf{z}) := \sum_{i \in [4]} \mathbf{z}[i] \cdot \text{Table}^{(k)}[i] = \text{Table}^{(k)}[(\bar{x}^{(i)}, \bar{x}^{(j)})], \quad (1)$$

where $\text{Table}^{(k)}$ is a masked truth table corresponding to the current gate, adjusted for the wire masks:

$$\text{Table}^{(k)}[(\bar{x}^{(i)}, \bar{x}^{(j)})] := g(\bar{x}^{(i)} \oplus r^{(i)}, \bar{x}^{(j)} \oplus r^{(j)}) \oplus r^{(k)} = \bar{x}^{(k)}.$$

The result is another authenticated share $\langle \mathbf{s} \cdot \bar{x}^{(k)} \rangle$.

3. The garbler reveals its share of $\langle \bar{x}^{(k)} \rangle \pmod{2}$ as part of the garbled circuit to the evaluator, who can recover the masked output $\bar{x}^{(k)}$. This step contributes to the 1-bit per gate garbling size.

We observe the most heavy computation by both the garbler and the evaluator are the HSS evaluation in step 2, in particular, evaluating the PRF inside HSS to derive the wire masks $r^{(i)}, r^{(j)}, r^{(k)}$.

Our scheme instead move the heavy PRF evaluations to a preprocessing stage. In more detail, both the garbler and evaluator initially hold FHE ciphertexts $\{\text{ct}_s(\text{key}[i])\}$, and homomorphically evaluate PRF over them via FHE in the preprocessing phase. They derive ciphertexts $\{\text{ct}_s(r^{(i)})\}$ of all wire masks $\{r^{(i)}\}$ after pre-processing. Assuming the derived FHE ciphertexts are compatible with HSS evaluations, the gate evaluation step 2 now proceeds as follows:

⁵ The distributed decryption algorithms use similar techniques to aHMAC evaluations in hindsight.

- 2'. Apply HSS evaluations between the authenticates shares $\langle \mathbf{s} \cdot \mathbf{z} \rangle$ and ciphertexts $\text{ct}_s(r^{(i)})$, $\text{ct}_s(r^{(j)})$, $\text{ct}_s(r^{(k)})$ to evaluate the function $f^{(k)}$ (Equation 1).

The HSS evaluation is now much simpler, consisting only of a constant number of HSS multiplications.

We emphasize that our modular separation of the garbling scheme into preprocessing and online phases allows separately optimizing each phase. The focus of this work is achieving a SIMD online phase, using our SIMD HSS and aHMAC presented earlier. Achieving SIMD in the offline phase is conceptually simple: just using a SIMD FHE scheme. A recent independent work [ZLY⁺26] explores this approach to improve concrete efficiency.

3 Preliminaries

Notations. We use bold letters $\mathbf{x}$ to denote a vector, and write $\mathbf{x}[i]$ to denote its i -th component. We write $\mathbf{x} \otimes \mathbf{y}$ to denote the tensor product between two vectors. For an integer $B > 0$, we write $[B]$ to denote the set $\{0, 1, \dots, B-1\}$. For an integer value within some range $x \in [B]$, we write $\text{Bits}(x)$ to denote its bit-representation as a Boolean vector of dimension $\lceil \log B \rceil$, and $\text{BitComp}(\mathbf{x} \in \{0, 1\}^{\lceil \log B \rceil})$ to denote the linear function that recovers x from its bit-representation.

When describing invocations of (sub-)protocols between two parties P_G, P_E , we write

$$(P_G : O_G), (P_E : O_E) \leftarrow \text{Protocol}((P_G : I_G), (P_E : I_E))$$

to mean the parties respectively hold inputs I_G, I_E when entering the protocol, and obtain outputs O_G, O_E after the protocol.

3.1 SIMD Computation

For every program $P : \mathcal{M}^{\ell_x} \rightarrow \mathcal{M}^{\ell_y}$ and integer $T \in \mathbb{Z}$, the T -SIMD version of P , denoted as $P^{\times T} : \mathcal{M}^{\ell_x \times T} \rightarrow \mathcal{M}^{\ell_y \times T}$ computing:

$$P^{\times T}(\{\mathbf{x}^{(t)}\}) := (P(\mathbf{x}^{(0)}), \dots, P(\mathbf{x}^{(T-1)}))$$

where $P(\mathbf{x}^{(t)})$ as the t 'th instance.

For a class of program $\mathcal{P}$, and a polynomial $T(\cdot)$, define $\mathcal{P}^{\times T} = \{\mathcal{P}^{T(\lambda)}\}_\lambda$ to be the class consisting of programs $P^{\times T(\lambda)}$ where $P \in \mathcal{P}$.

In this work, we will consider base programs that are integer arithmetic circuits. An arithmetic circuit C consists of addition and multiplication gates evaluated over $\mathbb{Z}$; denote by $\mathcal{C}$ the class of all integer arithmetic circuits. An arithmetic RMS program is a special case where multiplications are restricted to be between an intermediate wire value and an input value; denote by $\mathcal{P}_{\text{RMS}}$ the class of all integer RMS programs. Similar to prior works, schemes designed for these classes of programs only work when the inputs are B -admissible w.r.t. a known upper bound $B \in \mathbb{Z}$, i.e., $\text{adm}_B(C, \mathbf{x}) = 1$ iff all intermediate and input values are bounded by B .

3.2 Hard Problems

Definition 1 (RLWE). Let $\mathcal{R}(\lambda)$ be a ring, $q(\lambda)$ be a modulus, $\mathcal{D}_{\text{sk}}(\lambda), \mathcal{D}_{\text{err}}(\lambda) \subseteq \mathcal{R}(\lambda)$, be distributions for secrets and errors. The decisional ring learning with error (RLWE) problem

$\text{RLWE}_{\mathcal{R},q,\mathcal{D}_{\text{sk}},\mathcal{D}_{\text{err}}}$ is the following: distinguish $\{a_i, a_i \cdot s + e_i\}_{i \in [m]}$ and $\{a_i, b_i\}_{i \in [m]}$ for any polynomial $m(\lambda)$ number of samples with noticeable advantage, where $a_i \leftarrow_{\$} \mathcal{R}_q, s \leftarrow \mathcal{D}_{\text{sk}}, e_i \leftarrow \mathcal{D}_{\text{err}}$ and $b_i \leftarrow_{\$} \mathcal{R}_q$.

Definition 2 (PRLWE). Let $\mathcal{R}(\lambda)$ be a ring, $q(\lambda)$ be a modulus, $\mathcal{D}_{\text{sk}}(\lambda), \mathcal{D}_{\text{err}}(\lambda) \subseteq \mathcal{R}(\lambda)$, be distributions for secrets and errors. The decisional power ring learning with error (PRLWE) problem $\text{PRLWE}_{\mathcal{R},q,\mathcal{D}_{\text{sk}},\mathcal{D}_{\text{err}}}$ is the following: distinguish $(a_i, a_i \cdot s + e_{i,1}, s^2 \cdot a_i + e_{i,2})$ and (a_i, b_i, c_i) for any polynomial $m(\lambda)$ number of samples with noticeable advantage, where $a_i \leftarrow_{\$} \mathcal{R}_q, s \leftarrow \mathcal{D}_{\text{sk}}, e_{i,1}, e_{i,2} \leftarrow \mathcal{D}_{\text{err}}$ and $b_i, c_i \leftarrow_{\$} \mathcal{R}_q$.

Entropic secrets. We consider a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where n is a power-of-two, and require a specific secret distribution $\mathcal{D}_{\text{sk}}$ formally described in Figure 1. At a high level, for every secret $s \in \mathcal{D}_{\text{sk}}$, we require a fixed subset of coefficients in s to always be zero. We don't impose requirements on the remaining coefficients, except they should have bounded magnitude sufficiently smaller than the modulus q , as described in Figure 2.

We believe that this is a reasonable assumption since a similar requirement has been widely used in the FHE literature (e.g., [GHPS12, BCK⁺23, LW23, CKP25, LW24]), where the subset of low-degree coefficients is set to zero (e.g., imagine a ring element in $\mathcal{R} := \mathbb{Z}[X]/(X^n + 1)$ where only the first $n/2$ coefficients are short elements, while the second half being 0). This is assumed to be as secure as a normal secret, with an effective dimension equal to the number of non-zero coefficients.

Circular assumption. We additionally need circular assumptions, where the adversary is additionally given evaluations $\psi(s)$ of the secret s , masked by RLWE and PRLWE samples respectively. More formally, they are defined as follows:

Definition 3 (Circular RLWE). Let $\mathcal{R}(\lambda)$ be a ring, $q(\lambda)$ be a modulus, $\mathcal{D}_{\text{sk}}(\lambda), \mathcal{D}_{\text{err}}(\lambda) \subseteq \mathcal{R}(\lambda)$, be distributions for secrets and errors, and $\mathcal{F}(\lambda)$ be a set of functions $\psi : \mathcal{R}_q \rightarrow \mathcal{R}_q$. The circular RLWE problem $\text{cRLWE}_{\mathcal{R},q,\mathcal{D}_{\text{sk}},\mathcal{D}_{\text{err}},\mathcal{F}}$ is the following: distinguish $\{a_i, a_i \cdot s + e_i + \psi_i(s)\}_{i \in [m]}$ and $\{a_i, b_i\}_{i \in [m]}$ for any polynomial $m(\lambda)$ number of samples and sequence of functions $\{\psi_i \in \mathcal{F}\}_{i \in [m]}$, where $a_i \leftarrow_{\$} \mathcal{R}_q, s \leftarrow \mathcal{D}_{\text{sk}}, e_i \leftarrow \mathcal{D}_{\text{err}}$ and $b_i \leftarrow_{\$} \mathcal{R}_q$.

Definition 4 (Circular PRLWE). Let $\mathcal{R}(\lambda)$ be a ring, $q(\lambda)$ be a modulus, $\mathcal{D}_{\text{sk}}(\lambda), \mathcal{D}_{\text{err}}(\lambda) \subseteq \mathcal{R}(\lambda)$, be distributions for secrets and errors, and $\mathcal{F}(\lambda)$ be a set of functions $\psi : \mathcal{R}_q \rightarrow \mathcal{R}_q$. The circular PRLWE problem $\text{cPRLWE}_{\mathcal{R},q,\mathcal{D}_{\text{sk}},\mathcal{D}_{\text{err}},\mathcal{F}}$ is the following: distinguish $\{a_i, a_i \cdot s + e_{i,1} + \psi_{i,1}(s), s^2 \cdot a_i + e_{i,2} + \psi_{i,2}(s)\}_{i \in [m]}$ and $\{a_i, b_i, c_i\}_{i \in [m]}$ for any polynomial $m(\lambda)$ number of samples and sequence of functions $\{\psi_{i,1}, \psi_{i,2} \in \mathcal{F}\}_{i \in [m]}$, where $a_i \leftarrow_{\$} \mathcal{R}_q, s \leftarrow \mathcal{D}_{\text{sk}}, e_{i,1}, e_{i,2} \leftarrow \mathcal{D}_{\text{err}}$ and $b_i \leftarrow_{\$} \mathcal{R}_q$.

The specific functions ψ we consider are permutations of coefficients and scalar multiplication by a constant factor.

We note that circular assumptions are widely used in practice, especially in fully homomorphic encryption (FHE) [BGV14, Bra12, FV12, CKKS17, DM15, CGGI20] since we only know how to build FHE from circular assumptions.

Looking ahead, we will rely on entropic-circular-RLWE for type conversion and SIMD-HSS (Sections 4 and 5), and entropic-circular-PRLWE for SIMD-aHMAC and SIMD-Garbling (Section 5 and 6). As discussed in the introduction, both assumptions can be weakened to entropic-RLWE using existing techniques at the cost of increasing the size of public parameters by a depth-dependent term.

3.3 Fully Homomorphic Encryption

Definition 5 (Fully Homomorphic Encryption). A fully homomorphic encryption scheme consists of the following efficient algorithms:

- $\text{pk}, \text{evk}, \text{sk} \leftarrow \text{KeyGen}(\text{pp})$: which consists of the following two subroutines:
 - $\text{sk} \leftarrow \text{SKGen}(\text{pp})$: takes the public parameter pp ; outputs a secret key sk .
 - $\text{pk}, \text{evk} \leftarrow \text{PKGen}(\text{pp}, \text{sk})$: takes the public parameter pp , a secret key sk ; outputs a public key pair (pk, evk) .
 - $\text{KeyGen}(\text{pp}) := \text{Output } \text{pk}, \text{evk}, \text{sk} \text{ where } \text{sk} \leftarrow \text{SKGen}(\text{pp}), \text{pk}, \text{evk} \leftarrow \text{PKGen}(\text{pp}, \text{sk})$
- $\text{ct} \leftarrow \text{Enc}(\text{pp}, \text{pk}, m)$: takes the public parameter pp and the public key pk , generates the ciphertext c for the message m in plaintext space $\mathcal{P}$; outputs ct .
- $m \leftarrow \text{Dec}(\text{pp}, \text{sk}, \text{ct})$: takes the public parameter pp , decrypts the ciphertext ct into a plaintext message m based on the secret key sk ; outputs the message m .
- $\text{ct}'_1, \dots, \text{ct}'_{w'} \leftarrow \text{Eval}(\text{pp}, \text{evk}, f, \text{ct}_1, \dots, \text{ct}_w)$: takes the public parameter pp , an evaluation key evk , a function $f : \mathcal{P}^w \rightarrow \mathcal{P}^{w'}$, and ciphertexts $\text{ct}_1, \dots, \text{ct}_w$; output ciphertexts $\text{ct}'_1, \dots, \text{ct}'_{w'}$.

Correctness: For every polynomial $p(\lambda)$, let pp be public parameters dependent on λ , there exists a negligible function $\text{negl}(\lambda)$ such that: for all $\lambda \in \mathbb{N}$, and any function $f : \mathcal{P}^w \rightarrow \mathcal{P}^{w'}$ where $w, w' \leq p(\lambda)$, and inputs $m_0, \dots, m_w \in \mathcal{P}^w$, the following holds:

$$\Pr \left[\begin{array}{l} \text{Dec}(\text{pp}, \text{sk}, \text{ct}'_1), \dots, \text{Dec}(\text{pp}, \text{sk}, \text{ct}'_{w'}) \\ = f(m_1, \dots, m_w) \end{array} \middle| \begin{array}{l} (\text{pk}, \text{evk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp}) \\ \text{ct}_i \leftarrow \text{Enc}(\text{pp}, \text{pk}, m_i), \forall i \in [w] \\ \text{ct}'_0, \dots, \text{ct}'_{w'-1} \leftarrow \text{Eval}(\text{pp}, \text{evk}, f, \text{ct}_1, \dots, \text{ct}_w) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Security: Security is standard semantic security (IND-CPA security) of PKE schemes, so we omit the details.

Key switching We additionally key switching, which transforms a ciphertext under secret key sk_1 to a ciphertext under sk_2 without changing the underlying plaintext. To achieve this, we additionally define two algorithms $\text{KSKGen}(\text{pp}, \text{sk}_1, \text{sk}_2) \rightarrow \text{ksk}$ and $\text{KeySW}(\text{pp}, \text{ksk}, \text{ct}) \rightarrow \text{ct}'$. These two algorithms satisfy the following property: for any ciphertext $\text{ct} := (a, b) \in \mathcal{R}_q$ satisfying $b - a\text{sk}_1 = m \cdot \Delta + e$ where $m \in \mathcal{R}_t$ (for plaintext space $\mathcal{R}_t$ and $\Delta = \lceil q/t \rceil$) is a message and $\|e\| \leq B$ for some error bound $B \leq \Delta/2$, it holds that $\text{ct}' := (a', b') \in \mathcal{R}_q$ satisfies that $b' - a'\text{sk}_2 = m \cdot \Delta + e'$ for $\|e'\| \leq B_{\text{err}}$ for some $B \leq B_{\text{err}} \leq \Delta/2$. This is achievable in standard RLWE-based FHE systems [Bra12, FV12, BGV14, CKKS17, DM15, CGGI20].

In this work, we need something slightly stronger: let $\text{KSKGen}'(\text{pp}, \text{sk}_1, \text{sk}_2) \rightarrow \text{ksk}'$, $\text{KeySW}'(\text{pp}, \text{ksk}', \text{ct}) \rightarrow \text{ct}' := (\text{ct}_1 = (a_1, b_1), \text{ct}_2 = (a_2, b_2)) \in \mathcal{R}_q^4$, it holds that $b_1 - a_1\text{sk}_2 = \text{sk}_2 \cdot m \cdot \Delta + e'_1 \in \mathcal{R}_q$, and $b_2 - a_2\text{sk}_2 = m \cdot \Delta + e'_2 \in \mathcal{R}_q$, for $\|e'_1, e'_2\| \leq B_{\text{err}}$. We show below that this is indeed achievable.

Lemma 1. There exists an FHE scheme with the KSKGen' , KeySW' algorithms above with plaintext space $\mathcal{R}_t$ and public parameter $\text{pp} := (\mathcal{R}, q, t, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}})$, assuming the hardness of entropic RLWE $_{\mathcal{R}, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}}}$ (with $\mathcal{D}_{\text{sk}} \subseteq \mathcal{R}_t$)⁶ and circular security.

Proof sketch. Proof is simple: given RLWE with circular security, there exist FHE schemes with standard key switching [Bra12, FV12, BGV14, CGGI20]. Then, let $\text{sk}_i \leftarrow \text{SKGen}(\text{pp})$, we can define $\text{KSKGen}'(\text{pp}, \text{sk}_1, \text{sk}_2)$ as

⁶ We slightly abuse the notation here. $\mathcal{D}_{\text{sk}} \subseteq \mathcal{R}_t$ means that for all $x \leftarrow_{\$} \mathcal{D}_{\text{sk}}$, $x \in \mathcal{R}_t$ except for $\text{negl}(\lambda)$ probability.

- $\text{pk}_2, \text{evk}_2 \leftarrow \text{PKGen}(\text{pp}, \text{sk}_2)$
- $\text{ct}_{\text{sk}_2} \leftarrow \text{Enc}(\text{pk}_2, \text{sk}_2)$
- $\text{ksk} \leftarrow \text{KSKGen}(\text{pp}, \text{sk}_1, \text{sk}_2)$
- Output $\text{ksk}' := (\text{evk}_2, \text{ct}_{\text{sk}_2}, \text{ksk})$

Then, let $\text{ct} \leftarrow \text{Enc}(\text{pp}, \text{pk}_1, m)$ for any plaintext $m \in \mathcal{R}_t$, then, we can define $\text{KeySW}'(\text{pp}, \text{ksk}', \text{ct})$ as follows:

- $\text{ct}_2 \leftarrow \text{KeySW}(\text{pp}, \text{ksk}, \text{ct})$
- $\text{ct}_1 \leftarrow \text{Eval}(\text{pp}, \text{evk}_2, \times, \text{ct}_2, \text{ct}_{\text{sk}_2})$
- Output $\text{ct}' := (\text{ct}_1, \text{ct}_2)$

This achieves what we need in a straightforward way.

Note that here the keys have the following form: $\text{ksk} := (a_i, \text{sk}_2 a_i + e_i + \text{sk}_1 \cdot 2^i)$ for $i \in [\log(q)]$, $\text{evk}_2 := (a'_i, \text{sk}_2 a'_i + e'_i + \text{sk}_2^2 \cdot 2^i)$ for $i \in [\log(q)]$, $\text{pk}_2 := (a'', a'' \text{sk}_2 + e'')$, and $\text{ct}_{\text{sk}_2} := (a''', a''' \text{sk}_2 + e''' + \text{sk}_2 \cdot \Delta)$, where $a_i, a'_i, a'', a''', e_i, e'_i, e'', e''' \leftarrow_{\$} \mathcal{R}_q$. There are alternative ways to generate such keys [GHS12b], but since this is not the main focus of this work, for simplicity, we assume these keys have such forms. $\square$

Format of the key-switching key ksk' simply contains $\text{ksk}, \text{evk}_2, \text{ct}_{\text{sk}_2}$, which are RLWE ciphertexts (i.e., RLWE samples) encrypting $f_{\text{ksk}}(s_1)$ and $g_{\text{evk}}(s_2), g_{\text{Enc}}(\text{sk}_2)$ for some functions $f_{\text{ksk}}, g_{\text{evk}}, g_{\text{Enc}}$. By RLWE with circular assumption, these are indistinguishable from (a vector of) uniformly random $\mathcal{R}_q$ elements.

4 SIMD for Authenticated Shares from RLWE

Authenticated share [NNOB12] is an additive secret share of an authenticated value $\mathbf{s} \cdot x$ where $\mathbf{s} = (1, s) \in \mathcal{R}^2$ is a RLWE secret and x is a message. That is, we have shares $\langle x \rangle_0, \langle sx \rangle_0$ for one party and $\langle x \rangle_1, \langle sx \rangle_1$ for the other, so that $\langle x \rangle_1 - \langle x \rangle_0 = x$ and $\langle sx \rangle_1 - \langle sx \rangle_0 = sx$. For simplicity, we use the notation $\langle x \rangle$ and $\langle sx \rangle$ without specifying 0, 1 to denote both shares simultaneously. For a function $f(\langle \mathbf{s} \cdot x \rangle, \dots)$ on authenticated share(s), omitting $b = 0, 1$ refers to applying a function f on both shares. f_0 and f_1 denote local evaluations of f with respect to each party. If we use a different base ring other than $\mathcal{R}$, we specify it as a superscript (e.g. $\langle sx \rangle^{\mathcal{R}'}$). In this section, we introduce some gadgets/tools for authenticated shares that is used as key building blocks for SIMD computations in the later sections.

4.1 Interval Embeddings into $\mathcal{R}$

Let $S, T \in \mathbb{Z}_{>0}$ be integers such that $S \geq T$, S is a power-of-two, and $ST \leq n/2$. In our SIMD computation over $\mathcal{R}$, T is a SIMD factor and S is a parameter for clearing out some of the coefficients. Data is encrypted in the coefficients of polynomials in $\mathcal{R}$, and it is bounded by $B \in \mathbb{Z}_{>0}$ so that we support vectorized operations over $[B]^T$.

Given a vector $\mathbf{x} = (x_i)_i \in [B]^T$, we consider three types of encodings, namely **Type A**, **Type B**, and **Type C**, defined as

$$\mathcal{E}_{\mathbf{x}}^A := \sum_{i=0}^{T-1} x_i \cdot X^i \in \mathcal{R}, \quad \mathcal{E}_{\mathbf{x}}^B := \sum_{i=0}^{T-1} x_i \cdot X^{(S-1) \cdot i} \in \mathcal{R}, \quad \mathcal{E}_{\mathbf{x}}^C := \sum_{i=0}^{T-1} x_i \cdot X^{S \cdot i} \in \mathcal{R},$$

respectively. To understand the underlying intuition of these encodings, we consider two vectors $\mathbf{x}, \mathbf{y} \in [B]^T$ encoded in Type A and Type B, respectively. When we multiply $\mathcal{E}_{\mathbf{x}}^A$ and $\mathcal{E}_{\mathbf{y}}^B$ over $\mathcal{R}$, we get the following polynomial

$$\sum_{i=0}^{T-1} x_i \cdot X^i \times \sum_{i=0}^{T-1} y_i \cdot X^{(S-1) \cdot i} = \sum_{i=0}^{T-1} \sum_{j=0}^{T-1} (x_i y_j) \cdot X^{i+(S-1)j}.$$

Our key observation is that $i + (S-1)j$ is divisible by S if and only if $i = j$. Furthermore, the coefficient of the monomial $X^{S \cdot i}$ is exactly $x_i y_i$, meaning that these coefficients give a Hadamard multiplication of two vectors $\mathbf{x}$ and $\mathbf{y}$.

To write down this formally, let $\text{ClearCoeff}_S : \mathcal{R} \rightarrow \mathcal{R}$ be a function

$$\sum_{i=0}^{n-1} m_i \cdot X^i \mapsto \sum_{0 \leq i < n, S|i} m_i \cdot X^i$$

that zeros out all the coefficients other than whose degrees are multiple of S . Let $\mathcal{S} = \mathbb{Z}[X^S]/(X^n + 1) \subseteq \mathcal{R}$, then this function can be understood as a trace evaluation of $\mathcal{R}$ over $\mathcal{S}$. In particular, it can be efficiently instantiated with automorphisms over $\mathcal{R}$. The discussion above implies that

$$\mathcal{E}_{\mathbf{x} \odot \mathbf{y}}^C = \text{ClearCoeff}_S(\mathcal{E}_{\mathbf{x}}^A \cdot \mathcal{E}_{\mathbf{y}}^B),$$

where $\mathbf{x} \odot \mathbf{y}$ denotes the Hadamard product between $\mathbf{x}$ and $\mathbf{y}$. Note that all of the encodings are additive, directly giving SIMD additions. We write down the algorithms in Algorithm 1 and 2.

Algorithm 1 RingSIMDAdd($\mathcal{E}_{\mathbf{x}}^*, \mathcal{E}_{\mathbf{y}}^*$)

Input: $\mathbf{x}, \mathbf{y} \in [B]^T, \mathcal{E}_{\mathbf{x}}^*, \mathcal{E}_{\mathbf{y}}^* \in \mathcal{R}$ where $*$ $\in \{A, B, C\}$.

Output: $\mathcal{E}_{\mathbf{x}+\mathbf{y}}^*$.

return $\mathcal{E}_{\mathbf{x}}^* + \mathcal{E}_{\mathbf{y}}^*$

Algorithm 2 RingSIMDMult($\mathcal{E}_{\mathbf{x}}^A, \mathcal{E}_{\mathbf{y}}^B$)

Input: $\mathbf{x}, \mathbf{y} \in [B]^T, \mathcal{E}_{\mathbf{x}}^A, \mathcal{E}_{\mathbf{y}}^B \in \mathcal{R}$.

Output: $\mathcal{E}_{\mathbf{x} \odot \mathbf{y}}^C$, where $\mathbf{x} \odot \mathbf{y}$ denotes Hadamard multiplication.

return $\text{ClearCoeff}_S(\mathcal{E}_{\mathbf{x}}^A \cdot \mathcal{E}_{\mathbf{y}}^B)$

Although these operations already define SIMD addition and multiplication, it is not repeatable as input and output encoding types for multiplication are different. In the following section, we discuss how we define type conversions between different encodings, providing a complete framework for SIMD operations.

4.2 Authenticated Share Conversion over $\mathcal{R}$

We explore homomorphic evaluation of automorphisms on authenticated shares via HSS multiplication, and use it to define type conversions between different encoding types. These conversions are key ingredients of our SIMD HSS and aHMAC in the following sections.

Key Switching. Given an authenticated share $\langle \mathbf{s} \cdot x \rangle$ with a secret $\mathbf{s} = (1, s)$, key switching converts it to an authenticated share $\langle \mathbf{s}' \cdot x \rangle$ for another secret $\mathbf{s}' = (1, s')$. This is analogous to the key switching in the context of fully homomorphic encryption [Gen09]: We multiply an encryption of s under the secret s' .

Construction 1 (Authenticated Share Key Switching). The construction is with respect to the following public parameters $\mathbf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$:

- an input bound $B < 2^{\text{poly}(\lambda)}$;
- a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where $n(\lambda) < \text{poly}(\lambda)$ is a power-of-two;
- error and secret distribution $\mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}} \subseteq \mathcal{R}$ with coefficients bounded by $B_{\text{err}}, B_{\text{sk}} \leq \text{poly}(\lambda)$, respectively.
- two moduli p, q , where $q = \Delta \cdot p$ for a scaling factor Δ .
- A pseudo random function $\text{PRF} : \mathcal{K} \times \mathcal{S}_{\text{id}} \rightarrow \mathcal{R}$ where $\mathcal{K}$ and $\mathcal{S}_{\text{id}}$ are PRF key and operation id spaces, respectively. For simplicity, we omit it from the description of $\mathbf{pp}$, but implicitly assume its existence in the following functions.

$\text{evk} \leftarrow \text{KeyGen}(s, s')$: Given two secrets $s, s' \in \mathcal{R}$ sampled from $\mathcal{D}_{\text{sk}}$, the evaluation key evk is a RLWE encryption of s' under s . That is,

$$\text{evk} = (-as + \Delta \cdot s' + e, a)$$

where $a \leftarrow \mathcal{R}_q$ and $e \leftarrow \mathcal{D}_{\text{err}}$. We additionally generate a PRF key $K \leftarrow \mathcal{K}$ which is implicitly included in the evaluation key evk but omitted for simplicity.

$\langle \mathbf{s}' \cdot v \rangle \leftarrow \text{KeySwitch}(\text{evk}, \langle \mathbf{s} \cdot v \rangle)$: Given an authenticated share $\langle \mathbf{s} \cdot v \rangle$, an evaluation key evk , a PRF key $K \in \mathcal{K}$, and an operation id $\text{id} \in \mathcal{S}_{\text{id}}$, output an authenticated share $\langle \mathbf{s}' \cdot v \rangle$. This is instantiated via a multiplication between evk and $\langle \mathbf{s} \cdot v \rangle$:

$$(-as + \Delta \cdot s' + e, a) \cdot \langle (1, s) \cdot v \rangle = \langle (\Delta \cdot s' + e) \cdot v \rangle.$$

The distributed rounding is taken afterwards to remove the error term e and get $\langle s' \cdot v \rangle$. Since we already had $\langle v \rangle$, we have $\langle \mathbf{s} \cdot v \rangle$. Finally, we add a common pseudo-random shift to both parties' shares. This re-randomizes the share without affecting the shared value. The shift factor is obtained as $\text{PRF}(K, \text{id})$.

Lemma 2 (Key Switching Correctness). *The authenticated key switching in Construction 1 is correct up to a negligible failure probability $2^{-\kappa'}$ as long as it satisfies the following properties:*

$$p \geq n^2 \cdot B \cdot B_{\text{sk}} \cdot 2^{\kappa'} \text{ and } \Delta \geq n^2 \cdot B_{\text{err}} \cdot B \cdot 2^{\kappa'}. \quad (2)$$

Proof. We assume that PRF is evaluated every time we perform an operation so that input authenticated shares are always randomized. The distributed rounding happens once, when multiplying evk with $\langle (1, s) \cdot x \rangle$. Here, we have

$$(-as + \Delta \cdot s' + e) \cdot \langle x \rangle + a \cdot \langle sx \rangle = (s' \Delta + e) \cdot \langle x \rangle.$$

In other words, the two shares t_0, t_1 satisfy

$$t_1 - t_0 \equiv_q \Delta \cdot (s'x) + ex.$$

According to [BKS19, Lemma 1], the rounding failure probability is $n \cdot (\|ex\|_\infty + 1) \cdot p/q$ which has to be negligible ($\leq 2^{-\kappa'}$). According to [BKS19], the modulus lifting failure probability is $n \cdot (\|s'x\|_\infty + 1)/p$ which has to be negligible ($\leq 2^{-\kappa'}$). To summarize, it suffices to have

$$q \geq n^2 \cdot \|e\|_\infty \cdot B \cdot p \cdot 2^{\kappa'} \text{ and } p \geq n^2 \cdot \|s'\|_\infty \cdot B \cdot 2^{\kappa'}$$

which follows from the Equation 2. The final randomization from a PRF evaluation does not change the result. $\square$

Automorphism. Let $\varphi : \mathcal{R} \rightarrow \mathcal{R}$ be a ring automorphism. A homomorphic automorphism Aut converts an authenticated share $\langle \mathbf{s} \cdot v \rangle$ into an authenticated share $\langle \mathbf{s} \cdot \varphi(v) \rangle$. It can be instantiated with a key switching from s to $\varphi^{-1}(s)$.

Construction 2 (Automorphism for Authenticated Shares). Let the parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be the same as in Construction 1.

$\text{evk} \leftarrow \text{KeyGen}(s, \varphi^{-1}(s))$: Inherited from key switching, except that we convert from s to $\varphi^{-1}(s)$.
 $\langle \mathbf{s} \cdot \varphi(v) \rangle \leftarrow \text{Aut}^\varphi(\text{evk}, \langle \mathbf{s} \cdot v \rangle)$: Given $\langle \mathbf{s} \cdot v \rangle$, we first apply a key switching from s to $\varphi^{-1}(s)$ and get $\langle \varphi^{-1}(\mathbf{s}) \cdot v \rangle$. Next, we apply φ and get $\langle \mathbf{s} \cdot \varphi(v) \rangle$. Therefore,

$$\text{Aut}^\varphi(\text{evk}, \langle \mathbf{s} \cdot v \rangle) = \varphi \circ \text{KeySwitch}(\text{evk}, \langle \mathbf{s} \cdot v \rangle).$$

The correctness of automorphism automatically follows from Lemma 2.

Corollary 1 (Automorphism Correctness). *The automorphism for authenticated shares defined in Construction 2 is correct, as long as it satisfies the properties in Equation 2.*

Clearing Coefficients. Given a authenticated share $\langle \mathbf{s} \cdot v \rangle$ where $v \in \mathcal{R}$, we may use homomorphic automorphisms to zero out some of the coefficients. In particular, we consider removing all the coefficients other than whose degree are multiples of S . To be explicit, a polynomial $v = \sum_{i=0}^{n-1} v_i \cdot X^i$ is reduced to

$$\text{ClearCoeff}(v) := \sum_{0 \leq i < n, S|i} v_i \cdot X^i.$$

This operation can be efficiently instantiated with $\log S$ automorphisms.

Construction 3 (Clearing Coefficients). Let the parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be the same as in Construction 1.

$\text{evk} \leftarrow \text{TraceKeyGen}(s, S)$: Input a parameter S for clearing coefficients, the evaluation keys are generated by generating keys for the automorphisms $\varphi_i = X \mapsto X^{n/2^i+1}$ for each $0 \leq i < \log S$. That is,

$$\text{TraceKeyGen}(S) = (\text{KeyGen}(s, \varphi_i^{-1}(s)))_{0 \leq i < \log S}$$

$\langle \mathbf{s} \cdot v' \rangle \leftarrow \text{ClearCoeff}(\text{evk}, S, \langle \mathbf{s} \cdot v \rangle)$: Input an evaluation key evk generated via TraceKeyGen , the clearing coefficients parameter S , and an authenticated share $\langle \mathbf{s} \cdot v \rangle$, output an authenticated share $\langle \mathbf{s}' \cdot v' \rangle$ where $v' = \text{ClearCoeff}(v)$. This is instantiated via the algorithm in Algorithm 3.

Type A to B. Converting Type A to Type B simply requires substituting X with X^{S-1} . We observe that $\psi_S = X \mapsto X^{S-1}$ is an automorphism over $\mathcal{R}$, and a homomorphic evaluation of ψ_S directly gives a conversion.

Algorithm 3 ClearCoeff

Input: $u = \langle s \cdot v \rangle$, $\text{evk}_i = \text{KeyGen}(s, \varphi_i^{-1}(s))$ ($i \in [\log(S)]$), where $\varphi_i = X \mapsto X^{N/2^i+1}$.

Output: $\langle s \cdot \text{Tr}_{\mathcal{R}/S}(v)/2^S \rangle$.

for $i \leftarrow 0$ to $\log S - 1$ **do**

$u' \leftarrow \text{Aut}^{\varphi_i}(\text{evk}_i, u)$;

$u \leftarrow u + u'$;

end for

$u \leftarrow u/2^S$;

return u

Construction 4 (Type A to B Conversion). Let the parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be the same as in Construction 1. Let $\psi_S : \mathcal{R} \rightarrow \mathcal{R}$ be defined as $X \mapsto X^{S-1}$.

$\text{evk} \leftarrow \text{KeyGen}(s, \psi_S^{-1}(s))$: The evaluation key is generated by the switching key generation algorithm KeyGen with respect to secret keys s and $\psi_S^{-1}(s)$.

$\langle s \cdot \mathcal{E}_{\mathbf{x}}^B \rangle \leftarrow \text{ConvAB}(\text{evk}, \langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle)$: This is instantiated as

$$\text{ConvAB}(\text{evk}, v) = \text{Aut}^{\psi_S}(\text{evk}, v).$$

Given an input v encoding a vector $\mathbf{x}$ via Type A encoding, it outputs an authenticated share that encodes $\mathbf{x}$ via Type B encoding.

Theorem 4 (Type A to B Correctness). Let $\langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle$ be a pair of input authenticated shares that encrypts a vector $\mathbf{x} \in [B]^T$ via the Type A encoding. Let $\text{evk} = \text{KeyGen}(s, \psi_S^{-1}(s))$. Then we have

$$\langle s \cdot \mathcal{E}_{\mathbf{x}}^B \rangle = \text{Aut}^{\psi_S}(\text{evk}, \langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle).$$

Proof. It suffices to show that $\psi_S(\mathcal{E}_{\mathbf{x}}^A) = \mathcal{E}_{\mathbf{x}}^B$ which is true. The remaining part is checked by the correctness of the homomorphic automorphism. $\square$

Type C to A. In order to convert Type C to Type A, we need to substitute X^S with X . However, unlike Type A to Type B conversion, this is not an automorphism. To solve the problem, we ring switch to a larger ring where $X^S \mapsto X$ is an automorphism, evaluate this automorphism, and switch back to the original ring. To do this, we use a secret that is sparser than the binary uniform secret but still secure under entropic LWE assumption.

Construction 5 (Type C to A Conversion). Let the parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be as in Construction 1. Let $n' > n$ be an odd integer and $\mathcal{R}' = \mathbb{Z}[X]/(X^{n'} - 1)$. Let $\tilde{\psi} := X \mapsto X^S$ be the automorphism over $\mathcal{R}'$.

$\text{evk} \leftarrow \text{ConvCAKeyGen}$: It runs $\text{KeyGen}(s, s')$ where $s' = (\tilde{\psi}(s))_{\mathcal{R}}$.

$\langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle \leftarrow \text{ConvCA}(\text{evk}, \langle s \cdot \mathcal{E}_{\mathbf{x}}^C \rangle)$: Let $v = \mathcal{E}_{\mathbf{x}}^C$. The conversion is instantiated with the following step by step algorithms:

1. **Key Switching:** We key switch from s to s' , getting $\langle s' \cdot v \rangle^{\mathcal{R}}$.
2. **Embedding to $\mathbb{Z}[X]$:** Given an authenticated share pair $\langle s' \cdot v \rangle^{\mathcal{R}}$ (with $v = \mathcal{E}_{\mathbf{x}}^C$), we regard it as a share over $\mathbb{Z}[X]$.

$$\langle s' \cdot v \rangle^{\mathcal{R}} = \langle s' \cdot v \rangle^{\mathbb{Z}[X]} \quad [\text{when } \deg(s') + \deg(v) < n]$$

3. **Reduce it to $\mathcal{R}'$:** We regard $\langle s' \cdot v \rangle^{\mathbb{Z}[X]}$ as a valid element over $\mathcal{R}'$.

$$\langle s' \cdot v \rangle^{\mathbb{Z}[X]} = \langle s' \cdot v \rangle^{\mathcal{R}'}$$

4. **Automorphism over $\mathcal{R}'$:** Apply $\tilde{\psi}^{-1}$ on the share and get

$$\tilde{\psi}^{-1} \langle s' \cdot v \rangle^{\mathcal{R}'} = \langle \tilde{\psi}^{-1}(s' \cdot v)_{\mathcal{R}'} \rangle^{\mathcal{R}'} = \langle \tilde{\psi}^{-1}(\tilde{\psi}(s))_{\mathcal{R}'} \cdot \tilde{\psi}(v)_{\mathcal{R}'} \rangle^{\mathcal{R}'} = \langle s \cdot \tilde{\psi}(v) \rangle^{\mathcal{R}'}$$

5. **Embedding to $\mathbb{Z}[X]$:** We embed it to $\mathbb{Z}[X]$ and get

$$\langle s \cdot \tilde{\psi}(v)_{\mathcal{R}'} \rangle^{\mathbb{Z}[X]}. \quad [\text{when } \deg(s) + \deg(\tilde{\psi}(v)) < n'.]$$

6. **Reduce it to $\mathcal{R}$:** By reducing to $\mathcal{R}$, we get $\langle s \cdot \tilde{\psi}(v)_{\mathcal{R}'} \rangle^{\mathcal{R}} = \langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle$.

Theorem 5 (Type C to A Correctness). *Let $s \in \mathcal{R}$ be a secret key such that $\deg(\tilde{\psi}(s)) < n/2 + S$ and $\deg(s) < n' - T$. Then the conversion from Type C to A, defined in Construction 5 is correct.*

Proof. We check the correctness step by step.

1. The key switching correctness automatically checks the first step.
2. We check the embedding correctness. Let $w_i = \langle s' \cdot v \rangle_i^{\mathcal{R}}$ for $i = 0, 1$ so that $w_1 - w_0 \equiv_{\mathcal{R}} s' \cdot v$. Let us regard w_0, w_1 as elements of $\mathbb{Z}[X]$ instead of $\mathcal{R}$, by identifying $\mathcal{R}$ with polynomials of degree less than n . Since $w_1 - w_0 \equiv_{\mathcal{R}} s \cdot v$, we have $w_1 - w_0 = s \cdot v + e \cdot (X^n + 1)$ for some $e \in \mathbb{Z}[X]$. Since

$$\deg(s') + \deg(v) \leq \deg(s') + S(T - 1) < (n/2 + S) + (n/2 - S) = n,$$

we have that $\deg(s' \cdot v) < n$. Since

$$\deg(w_1 - w_0) \leq \max(\deg(w_1), \deg(w_2)) < n,$$

this implies that $e = 0$.

3. Reduction to $\mathcal{R}'$ is automatic, as one can simply take $[\cdot]_{\mathcal{R}'}$ on both shares.
4. As stated in the equation, $\tilde{\psi}$ and $\tilde{\psi}^{-1}$ cancels out and gets the correct result.
5. Similar to Step 1, it suffices to check that $\deg(s) + \deg(\tilde{\psi}(v)) < n'$. By the definition of $\tilde{\psi}$, we have that $\tilde{\psi}(\mathcal{E}_{\mathbf{x}}^C) = (\mathcal{E}_{\mathbf{x}}^A)_{\mathcal{R}'}$. Therefore,

$$\deg(s) + \deg(\tilde{\psi}(v)) < (n' - T) + T = n'.$$

6. Reducing to $\mathcal{R}$ is automatic, as one can simply take $[\cdot]_{\mathcal{R}}$ on both shares. We finally check that

$$\langle s \cdot \tilde{\psi}(v)_{\mathcal{R}'} \rangle^{\mathcal{R}} = \langle s \cdot (\mathcal{E}_{\mathbf{x}}^A)_{\mathcal{R}'} \rangle^{\mathcal{R}} = \langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle$$

as desired. This finishes the proof. □

In addition to SIMD addition and multiplication, our technique extends to other operations like packing, unpacking, rotation, and general computations. We detail the discussions in Appendix A.1.

Given a SIMD factor $T > 0$, a SIMD parameter $T \leq S \leq n/(2 \cdot T)$, and a ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ with a power-of-two dimension n , define:

- A helper ring $\mathcal{R}' = \mathbb{Z}[X]/(X^{n'} - 1)$ where $n' = n + 1$; ^a
- An automorphism (over $\mathcal{R}'$) $\tilde{\psi} = X \mapsto X^S : \mathcal{R}' \rightarrow \mathcal{R}'$.

We require a secret distribution $\mathcal{D}_{\text{sk}} \subseteq \mathcal{R}$ such that:

- For all $s \in \mathcal{D}_{\text{sk}}$, $\deg(\tilde{\psi}(s)) < n/2 + S$ and $\deg(s) < n' - T$.

For example, consider $S = T = \lfloor \sqrt{n/2} \rfloor$, and the subset $\text{bad} = \{i \in [n] : i \cdot S \bmod n' \geq n/2 + S\}$. We have $|\text{bad}| \leq n/2 - \sqrt{n/2} + 1$. We can choose $\mathcal{D}_{\text{sk}}$ to sample from independent discrete gaussian for every coefficient with index $< n' - T$ excluding those indexed by bad . (The rest are set to zero.) This results in at least $n/2$ non-zero coefficients.

^a More generally, n' can be any odd $n' > n$.

Fig. 1: Our requirement on RLWE secret distribution.

4.3 Our RLWE Assumption

At a high level, our SIMD type conversion algorithms impose two types of requirements on the underlying RLWE assumption.

- Correctness of the key-switching procedure requires a RLWE modulus $q = p \cdot \Delta$ composed from sufficiently large modulus p and scaling factor Δ .
- Correctness of our ring-switching procedure (used for Type-C-to-A conversions) require a sufficiently low degree on RLWE secrets s *after being transformed by the mapping* $\tilde{\psi} : X \mapsto X^S$. For a fixed ring $\mathcal{R}$ and SIMD parameter S , this translates to requiring a fixed subset of coefficients in s to zero.

Different applications considered in this work impose further requirements on the RLWE modulus, (see Figure 2) while the same degree requirement on RLWE (in Figure 1) secrets suffices for all of them.

4.4 Concrete Efficiency of SIMD Operations

Now, we move to analyze the concrete efficiency of these SIMD operations.

Unit operations. For our constructions, we need (1) ring additions, (2) ring multiplications, (3) division-and-rounding, (4) ring automorphisms, and (5) PRF calls. Since ring automorphisms (simply rearranging the ring coefficients) and PRF calls are essentially for free, we do not count these two operations.

One thing to note specifically is that for ring multiplications, instead of doing it naively, costing $\Omega(n^2)$ time (for ring with dimension n), we use the number theoretic transform (NTT) followed by element-wise multiplications, followed by inverse NTT (INTT), which costs $O(n \log(n))$ time. We call the ring element after NTT transformation to be in its “NTT” form, and before transformation to be in its “coefficient form” (see more discussion in Remark 1). Thus, when counting operations, we count the number of:

Given a SIMD factor $T > 0$, a SIMD parameter $T \leq S \leq n/(2 \cdot T)$, a ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ with a power-of-two dimension n , and magnitude bounds $B, B_{\text{err}}, B_{\text{sk}}$ on messages, RLWE errors and RLWE secrets, we require a RLWE modulus $q = p \cdot \Delta$ satisfying the following requirements, respectively for different applications:

SIMD-HSS: $p > \Omega(n^2 \cdot B \cdot B_{\text{sk}} \cdot \log n \cdot 2^{\kappa'})$; $\Delta > \Omega(n^2 \cdot B \cdot B_{\text{err}} \cdot \log n \cdot 2^{\kappa'})$.

SIMD-aHMAC: Same p as SIMD-HSS; $\Delta > \Omega(n^3 \cdot p \cdot B \cdot B_{\text{err}} \cdot \log n \cdot 2^{\kappa'})$.

SIMD-Garbling: Same p, Δ as SIMD-aHMAC, but with $B = 2$.^a

$2^{-\kappa'}$ represents the failure probability for a single multiplication in SIMD-HSS and SIMD-aHMAC, and a single Boolean gate in SIMD-Garbling. One should set $2^{-\kappa'}$ sufficiently small to account for a union bound over the entire computation.

^a Bounded integer arithmetics can emulate Boolean gates with $B = 2$.

Fig. 2: Our requirements on RLWE modulus.

- Ring addition, which can be done in both NTT and coefficient forms.
- NTT/INTT, transforming a ring element between NTT and coefficient forms.
- Element-wise multiplication, between two NTT form ring elements.
- Division-and-rounding (rounding for short) of a coefficient form ring element.

Remark 1 (Ring multiplication). To multiply two ring elements $x, y \in \mathcal{R}_q := \mathbb{Z}[X]/(X^n + 1)$, the most straightforward way is to compute xy being a degree $2n$ polynomial and then perform the modulo operation over $X^n + 1$. However, this process takes $O(n^2)$ time. Instead, one could use Number Theoretic Transform (NTT) and inverse NTT (INTT) to reduce this cost to $O(n \log(n))$.⁷ Specifically, $\text{NTT}(x) \rightarrow x'$ transforms a $\mathcal{R}_q$ element into a vector $x' \in \mathbb{Z}_q^n$. Note that this change simply change x from “coefficient” representation to “slot” representation. Then, $\text{INTT}(\text{NTT}(x)) \rightarrow x''$ is a symmetric operation of NTT that ensures that $x'' = x$. Both INTT and NTT take $O(n \log(n))$ time (and essentially the same amount of time since they are symmetric). Then, to compute $xy \in \mathcal{R}_q$, we can compute $\text{INTT}(\text{NTT}(x) \odot \text{NTT}(y))$ where $\odot$ is the element-wise multiplication. This process then takes $O(n \log(n))$ time.

More precisely, NTT/INTT takes $n \log(n)/2$ of $\mathbb{Z}_q$ multiplications while $\odot$ takes n such multiplications. Thus, to analyze our concrete efficiency, we count the number of NTTs needed and the number of element-wise multiplications. This is equivalent to counting the number of $\mathbb{Z}_q$ operations. We may additionally count the ring division and rounding operation. This takes n operations as well but in real numbers. In particular, for large q (e.g., $\gg 64$ bits), the rounding operation is costly due to the Chinese Remainder Theorem interpolation/reconstruction procedure, which we will discuss in more detail in Section 7.

Throughout the analysis, we omit the count of additions (more than $10\times$ faster in representative implementations, e.g., [BAB⁺22]), since additions are a lot faster than multiplications, while they have roughly the same amount. For simplicity, we call $\text{NTT}(x)$ the “NTT form” of x , which is in “coefficient form” to begin with.

⁷ For simplicity, let $\mathcal{R}$ fully split over q .

Key Switching. We start by analyzing the key switching operation in Lemma 2. It has input $\langle \mathbf{s} \cdot v \rangle = (\langle v \rangle, \langle \mathbf{s} \cdot v \rangle) \in \mathcal{R}_q^2$ (both in coefficient form), and $\text{evk} := (a, b) \in \mathcal{R}_q^2$ (in NTT form).⁸ Then, key switching multiplies $\langle \mathbf{s} \cdot v \rangle$ with evk followed by a rerandomization and a rounding-and-division operation, which takes 3 NTT/INTT transformations (2NTTs for the share and 1INTT to transform key switched share back to its coefficient form), 2 element-wise multiplications, 2 ring additions, and 1 rounding.

In some cases (e.g., for the `SIMDMultKey` for our aHMAC), $\langle v \rangle$ can be zero. With this constraint, we can save one NTT and one element-wise multiplication.

Automorphism. Automorphism has the same cost as key switching: recall that we ignore the runtime of ring element automorphism as discussed above. Note that the only automorphism we use is substitution, i.e., replacing X with some X^k , for which the automorphism of the ring element is self is almost free (since it only requires rearranging the coefficients, for both coefficient and NTT forms). Thus, one automorphism has the same cost as one key-switching.

Clearing Coefficients. The most costly process in this section is clearing coefficients `ClearCoeff` (Algorithm 3). It takes $\log(S)$ automorphisms, thus has the cost of $\log(S)$ key-switching. Looking ahead, we have $S = \sqrt{n/2}$ for a ring of dimension n . Thus, the cost of `ClearCoeff` is $\log(n/2)/2$ key-switching (for simplicity, we may write it as $\log(n)/2$).

Encoding Type Conversions. The conversion from type A to B requires only an automorphism, thus having the same cost as a single key-switching. The conversion from type C to A is slightly more complicated. Specifically, in addition to key-switching, it also requires the input shares to be in coefficient form to perform automorphism in a larger ring.

5 SIMD HSS and aHMAC

We use authenticated shares and the homomorphic evaluation procedures defined in Section 4 to construct HSS and aHMAC supporting efficient SIMD evaluations. Our constructions is built upon the RLWE-based HSS and aHMAC schemes by [BKS19, ILL25a] designed for (non-SIMD) evaluation of bounded integer arithmetic computation (RMS and circuits respectively). In these schemes, intermediate wire values are bounded integers, and encoded in separate authenticated shares over $\mathcal{R}$, leading to large shares per value and high computational cost.

Towards more efficient SIMD evaluations, we pack T values $\mathbf{x} \in [B]^T$ into a ring element using our special type-A, B, C encodings $\mathcal{E}_x^A, \mathcal{E}_x^B, \mathcal{E}_x^C$ described in Section 4, and hide them in a single authenticated share over $\mathcal{R}$. As shown in Table 1, SIMD multiplication over such SIMD encodings can be carried out using a few operations over the encodings, including ring multiplication, `ClearCoeff`, and `ConvCA`. Thus to have efficient SIMD multiplication, it boils down to designing efficient homomorphic evaluation algorithms for these three subroutines. Homomorphic procedure for `ClearCoeff`, and `ConvCA` are designed in the previous section. Below, we further observe that the homomorphic multiplication procedures in prior HSS and aHMAC [BKS19, ILL25a] already natively supports ring multiplication. Putting them together gives our new HSS and aHMAC schemes with efficient SIMD evaluations.

5.1 HSS Definitions

An HSS scheme consists of three algorithms `Setup`, `Input`, and `Eval` that's supposed to run by a trusted dealer and two parties P_0, P_1 . Initially, the dealer runs `Setup` to produce a public key

⁸ evk can be generated in NTT form during key generation.

pk , together with two (private) evaluation keys $\text{evk}_0, \text{evk}_1$, respectively for P_0 and P_1 . Then, the dealer runs **Input** to create secret shares $I_0^{(i)}, I_1^{(i)}$ of inputs x_i over some ring $\mathcal{M}$. Finally, each party P_b locally runs **Eval** over its input shares $\{I_b^{(i)}\}$ to homomorphically evaluate a program P . Correctness requires the evaluation results $\mathbf{z}_0, \mathbf{z}_1$ form a pair of additive secret shares of $\mathbf{z} = P(\mathbf{x})$. Security requires each party P_b learns no information about the shared inputs. We give the formal definition now.

Definition 6 (Homomorphic Secret Sharing (HSS)). *Let $\mathcal{P}$ be a class of program over a ring $\mathcal{M}$, and $\text{adm} = \{\text{adm}_\lambda\}_{\lambda \in \mathbb{Z}}$ a class of predicates where $\text{adm}_\lambda : \mathcal{P} \times \mathcal{M}^* \rightarrow \{0, 1\}$. A homomorphic secret sharing scheme for a class of programs $\mathcal{P}$ with admissible inputs defined by adm consists of three efficient algorithms:*

- **Setup**(1^λ) outputs a public key pk , and a pair of evaluation keys $\text{evk}_0, \text{evk}_1$.
- **Input**($\text{pk}, x \in \mathcal{M}$) takes as inputs the public key pk and an input x . It outputs a pair of input shares I_0, I_1 .
- **Eval**($b \in \{0, 1\}, \text{evk}_b, P \in \mathcal{P}, \mathbf{I}_b$) takes as inputs a party identity b , its evaluation key evk_b , a program $P : \{0, 1\}^{\ell_x} \rightarrow \{0, 1\}^{\ell_z}$, and input shares $\mathbf{I}_b$. It outputs its share of the evaluation results $\mathbf{z}_b \in \mathcal{M}^{\ell_z}$.

Correctness: *For all polynomial $p(\lambda)$, there exists a negligible function $\text{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, any program $P \in \mathcal{P}$ with ℓ_x inputs and ℓ_z outputs, and with size $|P| \leq p(\lambda)$, and any admissible inputs $\mathbf{x} \in \mathcal{M}^{\ell_x}$ such that $\text{adm}_\lambda(P, \mathbf{x}) = 1$, the following holds:*

$$\Pr \left[\begin{array}{c} \mathbf{z}_1 - \mathbf{z}_0 = P(\mathbf{x}) \\ \text{(over } \mathbb{Z}) \end{array} \middle| \begin{array}{l} (\text{pk}, \text{evk}_0, \text{evk}_1) \leftarrow \text{Setup}(1^\lambda) \\ (I_0^{(i)}, I_1^{(i)}) \leftarrow \text{Input}(\text{pk}, \mathbf{x}[i]) \\ \mathbf{I}_b := (\dots, I_b^{(i)}, \dots) \\ \mathbf{z}_b \leftarrow \text{Eval}(b, \text{evk}_b, P, \mathbf{I}_b) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Security: *For any efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$ and for all $b \in \{0, 1\}$:*

$$\left| \Pr[\text{Exp}_{\text{HSS}}^{\mathcal{A}, b, 0}(\lambda) = 1] - \Pr[\text{Exp}_{\text{HSS}}^{\mathcal{A}, b, 1}(\lambda) = 1] \right| \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}_{\text{HSS}}^{\mathcal{A}, b, \beta}$ is as follows:

1. Launch $\mathcal{A}(1^\lambda)$ and receive challenge inputs $x_0, x_1 \in \mathcal{M}$.
2. Run $(\text{pk}, \text{evk}_0, \text{evk}_1) \leftarrow \text{Setup}(1^\lambda)$ and $(I_0, I_1) \leftarrow \text{Input}(\text{pk}, x_\beta)$. Then send $(\text{pk}, \text{evk}_b, I_b)$ to $\mathcal{A}$.
3. In the end, $\mathcal{A}$ outputs a bit β' as the experiment result.

In this work, we will implement SIMD HSS for bounded integer RMS programs, meaning the scheme supports integer RMS programs $\mathcal{P}_{\text{RMS}}$ and inputs that are B -admissible for some apriori known upper bound B . The SIMD aspect means algorithms further takes in a SIMD factor T and evaluation is tailored to SIMD-programs $P^{\times T}$.

Definition 7 (SIMD HSS for Bounded-Integer RMS Programs). *A SIMD HSS scheme for bounded-integer RMS programs, is a tuple of algorithms $(\text{Setup}, \text{Input}, \text{Eval})$ such that for every efficiently computable polynomial $T : \mathbb{Z} \rightarrow \mathbb{Z}$ and exponential function $B : \mathbb{Z} \rightarrow \mathbb{Z}$ bounded by $2^{\text{poly}(\lambda)}$, the algorithms $(\text{Setup}^{T, B}, \text{Input}, \text{Eval})$ where $\text{Setup}^{T, B}(1^\lambda) = \text{Setup}(1^\lambda, 1^{T(\lambda)}, B(\lambda))$ is*

an HSS scheme (Definition 6) for the class of T -SIMD programs $\mathcal{P}_{RMS}^{\times T}$ with admissible inputs defined by $\{\text{adm}_{B(\lambda)}\}_{\lambda \in \mathbb{Z}}$.

To make explicit the SIMD syntax, we rewrite the Eval algorithm with the following interface: $\text{SIMDEval}(b, \text{evk}_b, P^{\times T}, \{\mathbf{I}_b^{(t)}\}_{t \in [T]})$, where $\mathbf{I}_b^{(t)}$ denotes the shares for input to the t -th instance.

5.2 SIMD HSS Multiplication

We first present the key component of our SIMD-HSS for bounded integer RMS programs – the SIMD homomorphic multiplication algorithm. As mentioned above, our construction uses the HSS construction [BKS19] based on RLWE to support homomorphic ring multiplications between a ciphertext (of appropriate form) and an authenticated share. We recall necessary preliminaries of [BKS19] below in Construction 6 and Lemma 3. Then, we describe in Construction 7 how to additionally apply the ClearCoeff and ConvCA algorithms to maintain proper SIMD encodings in our homomorphic multiplication algorithm.

Construction 6 (HSS Multiplication [BKS19]). The construction is with respect to the following public parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$:

- an input bound $B < 2^{\text{poly}(\lambda)}$;
- a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where $n(\lambda) < \text{poly}(\lambda)$ is a power-of-two;
- two modulus $p > B \cdot \lambda^{\omega(1)}$, and $q = p \cdot \Delta$, where $\Delta > B \cdot \lambda^{\omega(1)}$;
- error and secret distribution $\mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}} \subseteq \mathcal{R}$ with coefficients bounded by $\text{poly}(\lambda)$.

It uses a PRF $: \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \mathcal{R}_p$ as an ingredient.

In the following, we only recall the interfaces of Mult_b here, and refer readers to [BKS19] for construction details. We expose the details on PDGen, which is use in later constructions.

$\text{pd} \leftarrow \text{PDGen}(\text{pp})$: Outputs $\text{pd} = (\text{pp}, \text{seed})$, where $\text{seed} \leftarrow \{0, 1\}^\lambda$ is a PRF key.

$\langle \mathbf{s} \cdot z \rangle_b \leftarrow \text{Mult}_b(\text{pd}, \text{ct}_{s,x}, \langle \mathbf{s} \cdot y \rangle_b)$: takes public data pd , a ciphertext $\text{ct}_{s,x} \in \mathcal{R}_q^{2 \times 2}$, and the b -share of an authenticated share $\langle \mathbf{s} \cdot y \rangle_b$ w.r.t. some secret $\mathbf{s} \in \mathcal{R}^2$. It outputs the b -share of an authenticated share $\langle \mathbf{s} \cdot z \rangle_b$.

We have the following correctness lemma.

Lemma 3 (Correctness of HSS Multiplication [BKS19]). *Let $B_{\text{err}}(\lambda) < \text{poly}(\lambda)$ be an error bound. There exists a negligible function $\text{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, public parameters pp as specified, secret elements $s \in \mathcal{D}_{\text{sk}}$, message elements $x, y \in \mathcal{R}_B$ such that $\|xy\|_\infty < B$, authenticated shares (over $\mathcal{R}$) $\langle \mathbf{s} \cdot y \rangle_0, \langle \mathbf{s} \cdot y \rangle_1$ under $\mathbf{s} := (1, s)$, and ciphertexts $\text{ct}_{s,x} \in \mathcal{R}_q^{2 \times 2}$ satisfying the following decryption equation:*

$$\text{ct}_{s,x} \cdot \mathbf{s} = \mathbf{s} \cdot x \cdot \Delta + \mathbf{e}, \text{ for some } \|\mathbf{e}\|_\infty \leq B_{\text{err}},$$

the following holds:

$$\Pr \left[\begin{array}{c} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s} \cdot z \\ \text{(over } \mathcal{R}) \end{array} \middle| \begin{array}{c} \text{pd} \leftarrow \text{PDGen}(\text{pp}), \\ \mathbf{w}_b^z = \text{Mult}_b(\text{pd}, \text{ct}_{s,x}, \langle \mathbf{s} \cdot y \rangle_b) \\ z := xy \text{ (over } \mathcal{R}). \end{array} \right] > 1 - \text{negl}(\lambda),$$

Next, we present our SIMD homomorphic multiplication algorithms between a ciphertext $\text{ct}_{s,u}$ and an authenticated share $\langle \mathbf{s} \cdot v \rangle$, where $u = \mathcal{E}_{\mathbf{y}}^B \in \mathcal{R}$ should be a Type-B encoding of some vector $\mathbf{y} \in [B]^T$, and $v = \mathcal{E}_{\mathbf{x}}^A$ should be a Type-A encoding of some vector $\mathbf{x}$. The multiplication results should again form an authenticated share of a Type-A encoding $\mathcal{E}_{\mathbf{z}}^A$ of the SIMD product $\mathbf{z} = \mathbf{x} \odot \mathbf{y} \in [B]^T$.

Construction 7 (SIMD HSS Multiplication). The construction is with respect to the same public parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ as in Construction 6, except additionally the secret distribution $\mathcal{D}_{\text{sk}}$ should satisfy the requirement from Section 4.

The construction uses Construction 6, the ClearCoeff, ConvCA algorithms and their respective KeyGen algorithms as ingredients.

$\text{pd} \leftarrow \text{SIMDPDGen}(\text{pp}, s)$: Sample public evaluation keys for clearing coefficients and authenticated share conversions, and public data for HSS multiplication:

$$\begin{aligned} \text{CC.evk} &\leftarrow \text{TraceKeyGen}(s, T), \\ \text{CA.evk} &\leftarrow \text{ConvCAKeyGen}(s), \\ \text{HSS.pd} &\leftarrow \text{PDGen}(\text{pp}), \\ \text{pd} &:= (\text{CC.evk}, \text{CA.evk}, \text{HSS.pd}). \end{aligned} \tag{3}$$

$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^A \rangle_b \leftarrow \text{SIMDMult}_b(\text{pd}, \text{ct}_{s, \mathcal{E}_{\mathbf{x}}^B}, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_b)$: First parse $\text{pd} = (\text{CC.evk}, \text{CA.evk}, \text{HSS.pd})$, and run HSS multiplication over $\mathcal{R}$:

$$\langle \mathbf{s} \cdot w \rangle_b \leftarrow \text{HSS.Mult}_b(\text{HSS.pd}, \text{ct}_{s, \mathcal{E}_{\mathbf{x}}^B}, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_b),$$

where w is supposed to be the product of $\mathcal{E}_{\mathbf{x}}^A$ and $\mathcal{E}_{\mathbf{y}}^B$ over $\mathcal{R}$. In particular, the first T^2 coefficients of v should encode $\mathbf{x} \otimes \mathbf{y}$.

Next run ClearCoeff to set coefficients of v to zero, except those of $X^0, X^T, X^{2T}, \dots, X^{(T-1)T}$. The remaining coefficients should encode the SIMD product $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$ as a Type-C encoding.

$$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^C \rangle_b \leftarrow \text{ClearCoeff}_b(\text{CC.evk}, \langle \mathbf{s} \cdot w \rangle_b).$$

Finally, run ConvCA to convert the obtained authenticated share of Type-C encoding to a Type-A encoding.

$$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^A \rangle_b \leftarrow \text{ConvCA}_b(\text{CA.evk}, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^C \rangle_b).$$

We note the following correctness lemma, which follows directly from Lemma 3 and the correctness of ClearCoeff, and ConvCA.

Lemma 4 (Correctness of SIMD HSS Multiplication). *Let $B_{\text{err}}(\lambda) < \text{poly}(\lambda)$ be an error bound. There exists a negligible function $\text{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, public parameters pp as specified, secret elements $s \in \mathcal{D}_{\text{sk}}$, message vectors $\mathbf{x}, \mathbf{y} \in [B]^T$ such that their SIMD product $\mathbf{z} := \mathbf{x} \odot \mathbf{y}$ satisfy $\|\mathbf{z}\|_{\infty} < B$, authenticated shares (over $\mathcal{R}$) $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_0, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_1$ under $\mathbf{s} := (1, s)$, and ciphertexts $\text{ct}_{s, \mathcal{E}_{\mathbf{x}}^B} \in \mathcal{R}_q^{2 \times 2}$ satisfying the following decryption equation:*

$$\text{ct}_{s, \mathcal{E}_{\mathbf{x}}^B} \cdot \mathbf{s}^T = \mathbf{s}^T \cdot \mathcal{E}_{\mathbf{x}}^B \cdot \Delta + \mathbf{e}, \text{ for some } \|\mathbf{e}\|_{\infty} \leq B_{\text{err}},$$

the following holds:

$$\Pr \left[\begin{array}{l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}}^A \\ \text{(over } \mathcal{R}) \end{array} \middle| \begin{array}{l} \text{pd} \leftarrow \text{SIMDPDGen}(\text{pp}, s) \\ \mathbf{w}_b^z = \text{SIMDMult}_b(\text{pd}, \text{ct}_{s, \mathcal{E}_{\mathbf{x}}^B}, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_b) \\ \mathbf{z} := \mathbf{x} \odot \mathbf{y} \text{ (SIMD product over } \mathbb{Z}). \end{array} \right] > 1 - \text{negl}(\lambda),$$

5.3 SIMD HSS for Bounded Integer RMS

Now, we show in Construction 8 how to build a SIMD HSS scheme for bounded integer RMS programs as defined in Definition 7. We use the following representation of RMS programs.

Definition 8 (RMS Programs [Cle90, BGI16]). *The class of restricted multiplication straight-line (RMS) programs over integers consists of an arbitrary sequence of the following instructions, each with a unique identifier $\text{id} \in \mathbb{N}$:*

- (id, Load, i, j): Load an input $x^{(i)}$ into a memory value $y^{(j)}$;
- (id, Add, i, j, k): Add two memory values $y^{(i)}, y^{(j)}$ to obtain another $y^{(k)}$;
- (id, Mult, i, j, k): Multiply an input $x^{(i)}$ with a memory value $y^{(j)}$ to obtain a memory value $y^{(k)}$;
- (id, Out, i, j): Output a memory value $y^{(i)}$ as an output $z^{(j)}$.

Construction 8 (SIMD HSS for Bounded Integer RMS Programs). The construction uses Construction 7 (with public parameters specified in Setup) and a PRF : $\{0, 1\}^\lambda \times [2^\lambda] \rightarrow \{0, 1\}^\lambda$ as ingredients.

(pk, evk₀, evk₁) $\leftarrow$ Setup($1^\lambda, 1^T, B$): Let $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ be a polynomial ring where n is a power-of-two that's sufficiently large for SIMD encodings with factor T and interval $S = T$, as specified in Section 4. Let $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be public parameters specified in Figure 1 and 2.

- Prepare pk to contain an RLWE public element a and sample b :

$$\begin{aligned} s &\leftarrow \mathcal{D}_{\text{sk}}, a \leftarrow \mathcal{R}_q, e \leftarrow \mathcal{D}_{\text{err}}, \\ \text{pk} &:= (a, b), \text{ where } b \leftarrow s \cdot a + e. \end{aligned} \quad (4)$$

- Prepare evk₀, evk₁ to contain public data pd for SIMDMult and an authenticated share of $\mathcal{E}_1^A$ under $\mathbf{s} := (1, s)$.

$$\begin{aligned} \text{pd} &\leftarrow \text{SIMDPDGen}(\text{pp}, s), \\ \langle \mathbf{s} \cdot \mathcal{E}_1^A \rangle_0 &\leftarrow \mathcal{R}_p^2, \langle \mathbf{s} \cdot \mathcal{E}_1^A \rangle_1 \leftarrow \langle \mathbf{s} \rangle_0 + \mathbf{s} \cdot \mathcal{E}_1^A \text{ over } \mathcal{R}, \\ \text{evk}_b &:= (\text{pd}, \langle \mathbf{s} \cdot \mathcal{E}_1^A \rangle_b). \end{aligned}$$

(I₀, I₁) $\leftarrow$ Input(pk, x): Parse $\text{pk} = (a, b)$, and compute a ciphertext $\text{ct}_{s,x} \in \mathcal{R}_q^{2 \times 2}$ as follows. Output $I_0 = I_1 := \text{ct}_{s,x}$.

$$\begin{aligned} r_1, r_2 &\leftarrow \mathcal{D}_{\text{sk}}, e_1, e_2, e'_1, e'_2 \leftarrow \mathcal{D}_{\text{err}}, \\ \text{ct}_{s,x} &:= \begin{bmatrix} -c'_1 & c_1 \\ c'_2 & -c_2 \end{bmatrix}, \text{ where } \begin{aligned} c_1 &\leftarrow r_1 \cdot a + e_1, & c'_1 &\leftarrow r_1 \cdot b + e'_1 + x \cdot \Delta, \\ c_2 &\leftarrow r_2 \cdot a + e_2 + x \cdot \Delta, & c'_2 &\leftarrow r_2 \cdot b + e'_2, \end{aligned} \end{aligned} \quad (5)$$

$\mathbf{z}_b \leftarrow \text{SIMDEval}(b, \text{evk}_b, P, T, \{\mathbf{I}_b^{(t)}\}_{t \in [T]})$: First parse the input shares as $\mathbf{I}^{(t)}[i]_b = \text{ct}_{s,x(i,t)}$ for $i \in [\ell_x], t \in [T]$, and pack them into SIMD ciphertexts:

$$\text{ct}_{s, \mathcal{E}_{\mathbf{x}(i)}^B} \leftarrow \sum_{t \in [T]} \text{ct}_{s,x(i,t)} \cdot X^{(T-1) \cdot t}. \quad (6)$$

Next parse $\text{evk}_b = (\text{pd}, \langle \mathbf{s} \cdot \mathcal{E}_1^A \rangle_b)$, and read seed from pd. (See Construction 6.) For each of the id in P , prepare pd^{id} by replacing the seed in pd with $\text{seed}^{\text{id}} \leftarrow \text{PRF}(\text{seed}, \text{id})$. Finally process each instruction in P (over SIMD inputs and memory wires) as follows. We maintain the invariant that each SIMD memory wire corresponds to an authenticated share of a Type-A SIMD encoding.

(id, Load, i, j): Compute an authenticated share $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(j)}}^A \rangle_b$ corresponding to the j -th memory wire.

$$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(j)}}^A \rangle_b \leftarrow \text{SIMDMult}_b(\text{pd}^{\text{id}}, \text{ct}_{s, \mathcal{E}_{\mathbf{x}^{(i)}}^B}, \langle \mathbf{s} \cdot \mathcal{E}_1^A \rangle_b). \quad (7)$$

(id, Add, i, j, k): Let $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(i)}}^A \rangle_b, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(j)}}^A \rangle_b$ be the authenticated shares corresponding to the i, j -th memory wires, and compute

$$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(k)}}^A \rangle_b \leftarrow \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(i)}}^A \rangle_b + \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(j)}}^A \rangle_b \text{ over } \mathcal{R}. \quad (8)$$

(id, Mult, i, j, k): Let $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(j)}}^A \rangle_b$ be the authenticated share corresponding to the j -th memory wire and compute

$$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(k)}}^A \rangle_b \leftarrow \text{SIMDMult}_b(\text{pd}^{\text{id}}, \text{ct}_{s, \mathcal{E}_{\mathbf{x}^{(i)}}^B}, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(j)}}^A \rangle_b). \quad (9)$$

(id, Out, i, j): Let $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(i)}}^A \rangle_b$ be the authenticated share corresponding to the i -th memory wire. By definition, the share contains two elements over $\mathcal{R}$, we take the first T coefficients of the first element to be the output share $\mathbf{z}_b^{(j)} \in \mathbb{Z}^T$ of the j -th wire.

$$\mathbf{z}_b^{(j)} \leftarrow \text{Coeff}(\langle \mathcal{E}_{\mathbf{z}^{(j)}}^A \rangle_b)[1:T], \langle \mathcal{E}_{\mathbf{z}^{(j)}}^A \rangle_b \leftarrow \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{y}^{(i)}}^A \rangle_b[0]. \quad (10)$$

In the end, for the t -th SIMD instance of the program, collect the t -th component of the output shares on all output wires into a vector $\mathbf{z}_b^{(t)} \in \mathbb{Z}^{\ell_z}$. Output $\{\mathbf{z}_b^{(t)}\}_{t \in [T]}$.

Correctness: Let $\{\mathbf{x}^{(t)}\}_{t \in [T]}$ be inputs to a SIMD RMS program $P^{\times T}$. We first argue that the ciphertexts computed by Equation 5 and 6 encrypts type-B encodings of the SIMD inputs $\mathbf{x}^{(i)} = (\dots, x^{(i,t)}, \dots)_{t \in [T]}$, where $x^{(i,t)} := \mathbf{x}^{(t)}[i]$.

Claim. Let $p(\lambda), q(\lambda) \leq \text{poly}(\lambda)$ be the coefficient bound on distributions $\mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}}$, and n be the degree of $\mathcal{R}$. The ciphertexts $\text{ct}_{s, \mathcal{E}_{\mathbf{x}^{(i)}}^B}$ computed by Equation 5 and 6 satisfy

$$\text{ct}_{s, \mathcal{E}_{\mathbf{x}^{(i)}}^B} \cdot \mathbf{s}^\top = \mathbf{s}^\top \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^B \cdot \Delta + \mathbf{e}, \text{ for some } \|\mathbf{e}\|_\infty \leq B_{\text{err}} = O(t \cdot n \cdot p \cdot q),$$

where $\mathbf{s} := (1, s)$, and s is the secret sampled during Setup (Equation 4).

Proof. Write out the LHS according to Eq. 6:

$$\text{LHS} = \text{ct}_{s, \mathcal{E}_{\mathbf{x}^{(i)}}^B} \cdot \mathbf{s}^\top = \sum_t \underbrace{\text{ct}_{s, x^{(i,t)}} \cdot \mathbf{s}^\top}_{\text{res}^{(i,t)}} \cdot X^{(T-1) \cdot t}.$$

Then we examine $\text{res}^{(i,t)}$ according to Eq. 5:

$$\begin{aligned} \text{res}^{(i,t)} &= \begin{bmatrix} -c_1'^{(i,t)} + c_1^{(i,t)} \cdot s, \\ c_2'^{(i,t)} - c_2^{(i,t)} \cdot s \end{bmatrix} \\ (\text{Simplify}) &= \begin{bmatrix} r_1^{(i,t)} \\ -r_2^{(i,t)} \end{bmatrix} \cdot (b - s \cdot a) + \mathbf{s}^\top \cdot x^{(i,t)} \cdot \Delta + \mathbf{e}^{(i,t)} \\ &\quad // \|\mathbf{e}^{(i,t)}\|_\infty < O(n \cdot p \cdot q) \\ (\text{Eq. 4}) &= \mathbf{s}^\top \cdot x^{(i,t)} \cdot \Delta + \mathbf{e}'^{(i,t)} \quad // \|\mathbf{e}'^{(i,t)}\|_\infty < O(n \cdot p \cdot q) \end{aligned}$$

Now we can verify the claim:

$$\begin{aligned} LHS &= \sum_t \mathbf{s}^\top \cdot \left(x^{(i,t)} \cdot X^{(T-1) \cdot t} \right) \cdot \Delta + \sum_t \mathbf{e}'^{(i,t)} \cdot X^{(T-1) \cdot t} \\ &= \sum_t \mathbf{s}^\top \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^B \cdot \Delta + \mathbf{e} \quad // \|\mathbf{e}\|_\infty < O(t \cdot n \cdot p \cdot q) \end{aligned}$$

□

We next denote $\mathbf{y}^{(i)} = (\dots, y^{(i,t)}, \dots)_{t \in [T]}$ the correct j -th memory values when evaluating the SIMD program P . We argue the invariant that the **SIMDEval** algorithm correctly associates an authenticated share $\langle \mathbf{s}, \mathcal{E}_{\mathbf{y}^{(i)}}^A \rangle$ with every memory wire i . We examine the evaluation procedure for every gate inductively.

- (id, Load, i, j): By the above claim and Lemma 4, the output memory share computed by Equation 7 is an authenticated share of $\mathcal{E}_{\mathbf{x}^{(i)}}^A$.
- (id, Add, i, j, k): By assumption, the output memory share computed by Equation 8 is an authenticated share of $\mathcal{E}_{\mathbf{z}}^A$, where $\mathbf{z} = \mathbf{y}^{(i)} + \mathbf{y}^{(j)}$ over $\mathbb{Z}$.
- (id, Mult, i, j, k): Similarly by the above claim and Lemma 4, the output memory share computed by Equation 9 is an authenticated share of $\mathcal{E}_{\mathbf{z}}^A$ where $\mathbf{z} = \mathbf{y}^{(i)} \odot \mathbf{y}^{(j)}$.

Finally, we have verified every output wire j is associated with an authenticated share of the correct encoding $\mathcal{E}_{\mathbf{z}^{(j)}}^A$, we can conclude Equation 10 indeed computes an additive share of $\mathbf{z}^{(j)}$. Hence we have verified correctness.

Security: We prove security of Construction 8 and hence Theorem 6.

Theorem 6. *Construction 8 is a SIMD HSS scheme for bounded integer RMS programs per Definition 7, assuming RLWE with circular security with respect to $(\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}})$ as specified in Figure 1 and 2.*

Proof of Theorem 6. We have verified correctness above, and now focus on proving security. For any SIMD factor $T(\lambda) < \text{poly}(\lambda)$ input bound $B(\lambda) < 2^{\text{poly}(\lambda)}$, and a bit $b \in \{0, 1\}$, we show a series of hybrid experiments that transitions from $\text{Hyb}_0 = \text{Exp}_{\text{HSS}}^{\mathcal{A}, b, 0}(\lambda)$ to $\text{Hyb}_3 = \text{Exp}_{\text{HSS}}^{\mathcal{A}, b, 1}(\lambda)$.

Hyb₀: The experiment $\text{Exp}_{\text{HSS}}^{\mathcal{A}, b, 0}(\lambda)$, receives two inputs $x_0, x_1 \in \mathbb{Z}$ from $\mathcal{A}$, and sends $(\mathbf{pk}, \mathbf{evk}_b, I_b)$ to $\mathcal{A}$ computed as follows.

- Determine public parameters $\mathbf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ as in Setup.
- Compute $\mathbf{pk} = (a, b)$ as follows:

$$s \leftarrow \mathcal{D}_{\text{sk}}, a \leftarrow \mathcal{R}_q, e \leftarrow \mathcal{D}_{\text{err}}, b \leftarrow s \cdot a + e.$$

- Compute $I_b = (-c'_1, c_1, c'_2, -c_2)$ as follows:

$$\begin{aligned} r_1, r_2 &\leftarrow \mathcal{D}_{\text{sk}}, e_1, e_2, e'_1, e'_2 \leftarrow \mathcal{D}_{\text{err}}, \\ c_1 &\leftarrow r_1 \cdot a + e_1, c'_2 \leftarrow r_2 \cdot b + e'_2, \\ c'_1 &\leftarrow r_1 \cdot b + e'_1 + x_0 \cdot \Delta, c_2 \leftarrow r_2 \cdot a + e_2 + x_1 \cdot \Delta. \end{aligned}$$

– Prepare $\text{evk}_b = (\text{pd} := (\text{CC.evk}, \text{CA.evk}, \text{HSS.pd}), \mathbf{v}_b)$ as follows:

$$\begin{aligned} a^{(i)} &\leftarrow \mathcal{R}_q, e^{(i)} \leftarrow \mathcal{D}_{\text{err}}, b^{(i)} \leftarrow s \cdot a^{(i)} + e^{(i)} + \varphi^{(-1)}(s) \cdot \Delta, \\ &\text{for } i \in [\lceil \log T \rceil], \\ \text{CC.evk} &:= \{a^{(i)}, b^{(i)}\}_{i \in [\lceil \log T \rceil]}, \\ a' &\leftarrow \mathcal{R}_q, e' \leftarrow \mathcal{D}_{\text{err}}, b' \leftarrow s \cdot a' + e' + \varphi'(s) \cdot \Delta, \\ \text{CA.evk} &:= (a', b') \\ \text{HSS.pd} &:= (\text{pp}, \text{seed} \leftarrow \{0, 1\}^\lambda). \end{aligned}$$

where φ_i are ring automorphisms specified by `ClearCoeff`, and φ' is an one-to-one coefficient mapping specified by `ConvCA`. (See Section 4). Sample $\mathbf{v}_0 \leftarrow \mathcal{R}_q^2$ and set $\mathbf{v}_1 \leftarrow \mathbf{v}_0 + \mathbf{s} \cdot \mathcal{E}_1^A$.

Hyb₁: This hybrid samples b , $\{b^{(i)}\}$ and b' uniformly at random from $\mathcal{R}_q$, instead of computing them as RLWE with circular security samples. In particular, $I_b = (-c'_1, c_1, c'_2, -c_2)$ where $c'_1, c_1, c'_2, c_2 \leftarrow \mathcal{R}_q$, and $\text{pd} := (\text{CC.evk}, \text{CA.evk}, \text{HSS.pd})$ where

$$\begin{aligned} \text{CC.evk} &:= \{a^{(i)}, b^{(i)}\}_{i \in [\lceil \log T \rceil]}, \text{ for } a^{(i)}, b^{(i)} \leftarrow \mathcal{R}_q, \\ \text{CA.evk} &:= (a', b') \text{ for } a', b' \leftarrow \mathcal{R}_q, \\ \text{HSS.pd} &:= (\text{pp}, \text{seed} \leftarrow \{0, 1\}^\lambda). \end{aligned} \tag{11}$$

By our RLWE with circular security assumption, we have $\text{Hyb}_1 \approx_c \text{Hyb}_0$.

Hyb₂: This hybrid samples c_1, c_2, c'_1, c'_2 uniformly at random from $\mathcal{R}_q$, instead of computing them as RLWE ciphertexts encrypting x_0 . By standard RLWE (which is implied by our RLWE with circular security), we have $\text{Hyb}_2 \approx_c \text{Hyb}_1$.

Hyb₃: This hybrid runs the experiment $\text{Exp}_{\text{HSS}}^{A,b,1}(\lambda)$. By the same and symmetric arguments as the previous two hybrids, we have $\text{Hyb}_3 \approx_c \text{Hyb}_2$.

By a standard hybrid argument, we conclude $\text{Hyb}_0 = \text{Exp}_{\text{HSS}}^{A,b,0}(\lambda) \approx_c \text{Hyb}_3 = \text{Exp}_{\text{HSS}}^{A,b,1}(\lambda)$. $\square$

5.4 aHMAC Definitions

An aHMAC scheme consists of four algorithms `KeyGen`, `Input`, `EvalTag` and `EvalKey` that's supposed to run by two parties P_0, P_1 . Initially, the authenticator P_0 runs `KeyGen` to produce a secret key sk for itself and an evaluation key evk for P_1 . Then, P_0 runs `Auth` to create authentication tags $\sigma^{(i)}$ for inputs x_i over some ring $\mathcal{M}$ and sends them to P_1 . The evaluator P_1 can run `EvalTag` over authenticated inputs and tags to evaluate an arithmetic circuit C to derive a final tag σ^* that authenticates the result $C(\mathbf{x})$. In particular, σ^* should satisfy an algebraic format $\sigma^* = s^* \cdot C(\mathbf{x}) + k_C$, where s^* is chosen by P_0 during `KeyGen`, and k_C can be derived by P_0 by running `EvalKey` over C . Security requires that the global secret Δ chosen by P_0 remain hidden to P_1 . (See [ILL25a] for its connection to standard unforgeability security of homomorphic MACs.) We give the formal definition now.

Definition 9 (Algebraic Homomorphic MACs (aHMAC)). *let $\mathcal{C}$ be a class of arithmetic circuits over a ring $\mathcal{M}$, $\text{adm} = \{\text{adm}_\lambda\}_{\lambda \in \mathbb{Z}}$ a class of predicates where $\text{adm}_\lambda : \mathcal{C} \times \mathcal{M}^* \rightarrow \{0, 1\}$, and $\mathcal{S} \subseteq \mathcal{M}$ be a secret domain. An algebraic homomorphic MAC scheme for a class of arithmetic circuit $\mathcal{C}$ with admissible inputs defined by adm , and secret domain $\mathcal{S}$ consists of four efficient algorithms:*

- $\text{KeyGen}(1^\lambda, \mathbf{s}^* \in \mathcal{S}^{\ell_z})$ takes a vector of global secrets $\mathbf{s}^* \in \mathcal{S}^{\ell_z}$, (one for each evaluation output bit,) and outputs a secret key sk and evaluation key evk .
- $\text{Auth}(\text{sk}, x \in \mathcal{M}, \text{id} \in \{0, 1\}^\lambda)$ takes as inputs the secret key sk , a bit x to be authenticated, and an id associated with the bit. It outputs a tag σ_x . We will write $\sigma_{\mathbf{x}} = (\dots, \sigma_x^{(i)}, \dots)$ to mean a vector of such tags.
- $\text{EvalTag}(\text{evk}, C \in \mathcal{C}, \mathbf{x}, \sigma_{\mathbf{x}})$ takes as inputs the evaluation key evk , a circuit $C : \mathcal{M}^{\ell_x} \rightarrow \mathcal{M}^{\ell_z}$, input bits $\mathbf{x}$, and their associated tags $\sigma_{\mathbf{x}}$. It outputs tags $\sigma_{\mathbf{z}} \in \mathcal{M}^{\ell_z}$ authenticating the outputs of $C(\mathbf{x})$.
- $\text{EvalKey}(\text{sk}, C, \mathbf{id})$ takes as inputs the secret key sk , a circuit C and the ids associated with its inputs. It outputs MAC keys $\mathbf{k}_C \in \mathcal{M}^{\ell_z}$ for the outputs of C .

Correctness: For every polynomial $p(\lambda)$, there exists a negligible function $\text{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, arithmetic circuits $C \in \mathcal{C}$, where $C : \mathcal{M}^{\ell_x} \rightarrow \mathcal{M}^{\ell_z}$ and $|C| \leq p(\lambda)$, global secrets $\mathbf{s}^* \in \mathcal{S}^{\ell_z}$, admissible inputs $\mathbf{x} \in \mathcal{M}^{\ell_x}$ such that $\text{adm}_\lambda(C, \mathbf{x}) = 1$, and $\text{ids } \mathbf{id} \in \{0, 1\}^{\ell_x \times \lambda}$, the following holds:

$$\Pr \left[\begin{array}{l} \sigma_{\mathbf{z}} = \mathbf{s}^* \odot C(\mathbf{x}) + \mathbf{k}_C \\ \text{(over } \mathcal{M}^{\ell_z}, \odot \text{ means} \\ \text{component-wise mult)} \end{array} \middle| \begin{array}{l} (\text{sk}, \text{evk}) \leftarrow \text{KeyGen}(1^\lambda, \mathbf{s}^*) \\ \sigma^{(i)} \leftarrow \text{Auth}(\text{sk}, \mathbf{x}[i], \mathbf{id}[i]) \\ \sigma_{\mathbf{x}} := (\sigma^{(0)}, \dots, \sigma^{(\ell_x-1)}) \\ \sigma_{\mathbf{z}} \leftarrow \text{EvalTag}(\text{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \\ \mathbf{k}_C \leftarrow \text{EvalKey}(\text{sk}, C, \mathbf{id}) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Security: There exists a pair of efficient simulators $\text{Sim}_1, \text{Sim}_2$ such that for every efficient adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\left| \Pr[\text{Exp}_{\text{priv}}^{\mathcal{A},0}(\lambda) = 1] - \Pr[\text{Exp}_{\text{priv}}^{\mathcal{A},1}(\lambda) = 1] \right| \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}_{\text{priv}}^{\mathcal{A},b}$ is as follows:

1. Launch $\mathcal{A}(1^\lambda)$. Receive from $\mathcal{A}$ a vector $\mathbf{s}^* \in \mathcal{M}^{\ell_z}$, compute evk as follows and send it to $\mathcal{A}$.

$$\begin{cases} \text{evk}, \text{sk} \leftarrow \text{KeyGen}(1^\lambda, \mathbf{s}^*) & \text{if } b = 0, \\ \text{evk}, \text{st} \leftarrow \text{Sim}_1(1^{\ell_z}) & \text{if } b = 1, \end{cases}$$

2. Receive any number of adaptively chosen queries $(x \in \mathcal{M}, \text{id} \in \{0, 1\}^\lambda)$ with distinct ids from $\mathcal{A}$, and answer each with a tag σ computed as follows.

$$\begin{cases} \sigma \leftarrow \text{Auth}(\text{sk}, x, \text{id}) & \text{if } b = 0, \\ \sigma, \text{st} \leftarrow \text{Sim}_2(\text{st}) & \text{if } b = 1. \end{cases}$$

(Queries with previously used ids are ignored.)

3. $\mathcal{A}$ outputs a bit b' as the output of the experiment.

Succinctness: The tags produced by Auth and EvalTag have bounded sizes by some fixed $\text{poly}(\lambda)$.

In this work, we will implement SIMD HSS for bounded integer arithmetic circuits, meaning the scheme supports arithmetic circuits $\mathcal{C}_{\text{Arith}}$ and integer inputs that are B -admissible for some apriori known upper bound B . The SIMD aspect means algorithms further takes in a SIMD factor T and evaluation is tailored to SIMD-programs $C^{\times T}$.

Definition 10 (SIMD aHMAC for Bounded Integer Arithmetic Circuits). A SIMD HSS scheme for bounded-integer arithmetic circuits, is a tuple of algorithms $(\text{KeyGen}, \text{Auth}, \text{EvalTag}, \text{EvalKey})$ such that for every efficiently computable polynomial $T : \mathbb{Z} \rightarrow \mathbb{Z}$ and exponential function $B : \mathbb{Z} \rightarrow \mathbb{Z}$ bounded by $2^{\text{poly}(\lambda)}$, the algorithms $(\text{KeyGen}^{T,B}, \text{Auth}, \text{EvalTag}, \text{EvalKey})$ where $\text{KeyGen}^{T,B}(1^\lambda, \mathbf{s}^*) = \text{KeyGen}(1^\lambda, 1^{T(\lambda)}, B(\lambda), \mathbf{s}^*)$ is an aHMAC scheme (Definition 9) for the class of T -SIMD circuits $\mathcal{C}_{\text{Arith}}^{\times T}$ with admissible inputs defined by $\{\text{adm}_{B(\lambda)}\}_{\lambda \in \mathbb{Z}}$, and secret domain $\mathcal{S} = [2^\lambda]$.

To make explicit the SIMD syntax, we rewrite the KeyGen , EvalTag , and EvalKey algorithms with the following interfaces: $\text{KeyGen}(1^\lambda, 1^T, B, \{\mathbf{s}^{*,(t)}\}_{t \in [T]})$, $\text{EvalTag}(\text{evk}, C^{\times T}, \{\mathbf{x}^{(t)}\}, \{\sigma_{\mathbf{x}}^{(t)}\})$, and $\text{EvalKey}(\text{sk}, C^{\times T}, \{\text{id}^{(t)}\})$, where the superscripts (t) denote the t -th instance.

5.5 SIMD aHMAC Multiplication

Similarly to our SIMD-HSS construction, we first present a key component which is a pair of SIMD homomorphic multiplication algorithms on authentication keys and tags. Due to their algebraic formats in our constructions, a pair of key and tag of some value x can be viewed as a pair of authenticated shares $\langle s \cdot x \rangle_0, \langle s \cdot x \rangle_1$ with respect to some secret s . We explicitly write them as shares in the following to emphasize their algebraic properties.

As mentioned above, our construction uses the aHMAC construction [ILL25b] based on RLWE to implement multiplications of tags and keys of polynomial ring $\mathcal{R}$ elements. We recall necessary preliminaries below in Construction 9 and Lemma 5. Then, we describe in Construction 10 how to additionally apply the ClearCoeff, ConvAB and ConvCA algorithms to maintain proper SIMD encodings in our authentication tags and keys.

Construction 9 (aHMAC Multiplication [ILL25b]). The construction is with respect to the same public parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ as Construction 6, except the scaling factor $\Delta = q/p$ additionally needs to satisfy $\Delta > B \cdot p \cdot \lambda^{\omega(1)}$ (instead of $\Delta > B \cdot \lambda^{\omega(1)}$). It uses a PRF $: \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \mathcal{R}_p$ as an ingradient.

In the following, we only recall the interfaces of MultTag , MultKey here, and refer readers to [ILL25b] for construction details. We expose the details on PDGen , which is use in later constructions.

$\text{PDGen}(\text{pp}, s)$: takes a secret $s \in \mathcal{D}_{\text{sk}}$, and computes pd as follows.

$$\begin{aligned} a &\leftarrow \mathcal{R}_q, e_1, e_2 \leftarrow \mathcal{D}_{\text{err}}, \text{seed} \leftarrow \{0, 1\}^\lambda, \\ b &\leftarrow s \cdot a + e_1, c \leftarrow s^2 \cdot a + e_2 - s \cdot \Delta, \\ \text{pd} &:= (\text{pp}, \text{seed}, a, b, c). \end{aligned}$$

$\text{MultKey}(\text{pd}, \langle s \cdot x \rangle_0, \langle s \cdot y \rangle_0)$: takes as inputs public data pd and two ring elements representing the 0-shares of some authenticated shares $\langle s \cdot x \rangle_0, \langle s \cdot y \rangle_0 \in \mathcal{R}$ w.r.t. some secrets $s \in \mathcal{R}$. It outputs the 0-share of an authenticated share $\langle s \cdot z \rangle_0$.

$\text{MultTag}(\text{pd}, \langle s \cdot x \rangle_1, \langle s \cdot y \rangle_1, x, y)$: takes as input public data pd , the 1-shares of some authenticated shares $\langle s \cdot x \rangle_1, \langle s \cdot y \rangle_1$ w.r.t. some secret s , and the shared values $x, y \in \mathcal{R}$. It outputs the 1-share of an authenticated share $\langle s \cdot z \rangle_1$.

Lemma 5 (Correctness of aHMAC Multiplication [ILL25a]). There exists a negligible function $\text{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, public parameters pp as specified, secret elements $s \in \mathcal{D}_{\text{sk}}$, inputs $x, y \in \mathcal{R}_B$ such that $\|xy\|_\infty < B$, and additive shares (over $\mathcal{R}$)

$\langle sx \rangle_0, \langle sx \rangle_1, \langle sy \rangle_0, \langle sy \rangle_1$, the following holds:

$$\Pr \left[\begin{array}{c} w_1 = w_0 + s \cdot xy \\ (\text{over } \mathcal{R}) \end{array} \middle| \begin{array}{l} \text{pd} \leftarrow \text{PDGen}(\text{pp}, s) \\ w_0 = \text{MultKey}(\text{pd}, \langle sx \rangle_0, \langle sy \rangle_0) \\ w_1 = \text{MultTag}(\text{pd}, \langle sx \rangle_1, \langle sy \rangle_1, x, y) \end{array} \right] > 1 - \text{negl}(\lambda).$$

Next, we present our SIMD multiplication algorithms for authentication tags and keys of Type-A encodings $\mathcal{E}_x^A, \mathcal{E}_y^B \in \mathcal{R}$ of some vectors $\mathbf{x}, \mathbf{y} \in [B]^T$. The multiplication results should again form a pair of tag and key of a Type-A encoding $\mathcal{E}_z^A$ of the SIMD product $\mathbf{z} = \mathbf{x} \odot \mathbf{y} \in [B]^T$.

Construction 10 (SIMD aHMAC Multiplication). The construction is with respect to the same public parameters pp as in Construction 9, except additionally the secret distribution $\mathcal{D}_{\text{sk}}$ should satisfy the requirement from Section 4.

The construction uses Construction 9, the ClearCoeff, ConvCA, ConvAB algorithms and their respective KeyGen algorithms ingredients.

$\text{pd} \leftarrow \text{SIMDPDGen}(\text{pp}, s)$: Sample public evaluation keys for clearing coefficients and authenticated share conversions, and sample public data for aHMAC multiplications:

$$\begin{aligned} \text{CC.evk} &\leftarrow \text{TraceKeyGen}(s, T), \\ \text{CA.evk} &\leftarrow \text{ConvCAKeyGen}(s), \text{AB.evk} \leftarrow \text{ConvABKeyGen}(s), \\ \text{aHMAC.pd} &\leftarrow \text{PDGen}(\text{pp}, s). \end{aligned} \tag{12}$$

$\langle s \cdot \mathcal{E}_z^A \rangle_0 \leftarrow \text{SIMDMultKey}(\text{pd}, \langle s \cdot \mathcal{E}_x^A \rangle_0, \langle s \cdot \mathcal{E}_y^A \rangle_0)$: first convert the left multiplicand from a Type-A to a Type-B encoding.

$$\langle s \cdot \mathcal{E}_x^B \rangle_0 \leftarrow \text{ConvAB}(\text{AB.evk}, \langle s \cdot \mathcal{E}_x^A \rangle_0).$$

Then, analogously to our SIMD HSS construction (Construction 7), run aHMAC mult to homomorphically multiply $\mathcal{E}_x^B$ and $\mathcal{E}_y^A$, and run ClearCoeff to derive a Type-C encoding of $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$.

$$\begin{aligned} \langle s \cdot w \rangle_0 &\leftarrow \text{MultKey}(\text{aHMAC.pd}, \langle s \cdot \mathcal{E}_x^B \rangle_0, \langle s \cdot \mathcal{E}_y^A \rangle_0), \\ \langle s \cdot \mathcal{E}_z^C \rangle_0 &\leftarrow \text{ClearCoeff}_0(\text{CC.evk}, \langle s \cdot w \rangle_0), \text{ where } \mathbf{s} := (1, s), \text{ and } \langle w \rangle_0 \leftarrow 0. \end{aligned} \tag{13}$$

Finally, run ConvCA to convert the authenticated share of a Type-C encoding to Type-A.

$$\begin{aligned} \langle s \cdot \mathcal{E}_z^A \rangle_0 &\leftarrow \text{ConvCA}_0(\text{CA.evk}, \langle s \cdot \mathcal{E}_z^C \rangle_0), \\ \langle s \cdot \mathcal{E}_z^A \rangle_0 &\leftarrow \langle s \cdot \mathcal{E}_z^A \rangle_0[1]. \end{aligned}$$

$\langle s \cdot \mathcal{E}_z^A \rangle_1 \leftarrow \text{SIMDMultTag}(\text{pd}, \langle s \cdot \mathcal{E}_x^A \rangle_1, \langle s \cdot \mathcal{E}_y^A \rangle_1, \mathcal{E}_x^A, \mathcal{E}_y^A)$: Analogous to SIMDMultKey, except the second step (Equation 13) differs as follows.

$$\begin{aligned} \langle s \cdot w \rangle_1 &\leftarrow \text{MultTag}(\text{aHMAC.pd}, \langle s \cdot \mathcal{E}_x^B \rangle_1, \langle s \cdot \mathcal{E}_y^A \rangle_1, \boxed{\mathcal{E}_x^B, \mathcal{E}_y^A}), \\ \langle s \cdot \mathcal{E}_z^C \rangle_1 &\leftarrow \text{ClearCoeff}_1(\text{CC.evk}, \langle s \cdot w \rangle_1), \text{ where } \mathbf{s} := (1, s), \text{ and } \boxed{\langle w \rangle_1 \leftarrow w}. \end{aligned}$$

where $\mathcal{E}_x^B, \mathcal{E}_y^A$ and $w := \mathcal{E}_x^B \cdot \mathcal{E}_y^A$ over $\mathcal{R}$ can be derived from the inputs $\mathcal{E}_x^A, \mathcal{E}_y^A$.

We note the following correctness lemma, which follows from Lemma 5 and the correctness of ClearCoeff, ConvCA, ConvAB.

Lemma 6 (Correctness of SIMD aHMAC Multiplication). *There exists a negligible function $\text{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, public parameters pp as specified, secret elements $s \in \mathcal{D}_{\text{sk}}$, inputs $\mathbf{x}, \mathbf{y} \in [B]^T$ such that their SIMD product $\mathbf{z} := \mathbf{x} \odot \mathbf{y}$ satisfy $\|\mathbf{z}\|_\infty < B$, and additive shares (over $\mathcal{R}$) $\langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle_0, \langle s \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_0, \langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle_1, \langle s \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_1$, the following holds:*

$$\Pr \left[\begin{array}{l} w_1 = w_0 + s \cdot \mathcal{E}_{\mathbf{x} \odot \mathbf{y}}^A \\ \text{(over } \mathcal{R}) \end{array} \middle| \begin{array}{l} \text{pd} \leftarrow \text{SIMDPDGen}(\text{pp}, s), \\ w_0 = \text{SIMDMultKey}(\text{pd}, \langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle_0, \langle s \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_0), \\ w_1 = \text{SIMDMultTag}(\text{pd}, \langle s \cdot \mathcal{E}_{\mathbf{x}}^A \rangle_1, \langle s \cdot \mathcal{E}_{\mathbf{y}}^A \rangle_1, \mathcal{E}_{\mathbf{x}}^A, \mathcal{E}_{\mathbf{y}}^A) \end{array} \right] > 1 - \text{negl}(\lambda).$$

5.6 SIMD aHMAC for Bounded Integer Arithmetic Circuits

Now, we show in Construction 11 how to build a SIMD aHMAC scheme for B -bounded integer arithmetic circuits with secret domain $\mathcal{S} = [2^\lambda]$ as defined in Definition 10.

Construction 11 (SIMD aHMAC for Bounded Integer Arithmetic Circuits). The construction uses Construction 10 (with public parameters specified in KeyGen), $\text{PRF} : \{0, 1\}^\lambda \times [2^\lambda] \rightarrow \{0, 1\}^\lambda$, $\text{PRF}' : \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \mathcal{R}_p$, the KeySW, UnPack algorithms and their corresponding KeyGen algorithm as ingredients.

$(\text{evk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda, 1^T, B, \{\mathbf{s}^{*,(t)}\}_{t \in [T]})$: Let $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ be a polynomial ring where n is a power-of-two sufficiently large for SIMD encodings with factor T and interval $S = T$ as specified in Section 4. Let $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be public parameters specified in Figure 1 and 2.

- Prepare evaluation keys for KeySW, UnPack, and public data for SIMD aHMAC multiplication:

$$\begin{aligned} s &\leftarrow \mathcal{D}_{\text{sk}}, \text{UP.evk} \leftarrow \text{UP.KeyGen}(s) \\ \text{KS.evk}^{(i,j,t)} &\leftarrow \text{KS.KeyGen}(s, \text{Bits}(\mathbf{s}^{*,(t)}[i])[j]) \text{ for } t \in [T], i \in [\ell_z], j \in [\lambda], \\ \text{pd} &\leftarrow \text{SIMDPDGen}(\text{pp}, s). \end{aligned}$$

- Sample two PRF keys $\text{key}, \text{key}' \leftarrow \{0, 1\}^\lambda$, and outputs

$$\text{evk} := (\text{key}, \text{pd}, \text{UP.evk}, \{\text{KS.evk}^{(i,j,t)}\}), \text{sk} := (\text{key}', \text{evk}).$$

$\sigma_x \leftarrow \text{Auth}(\text{sk}, x, \text{id})$: (Unchanged from [ILL25a].) Read s, key' from sk and compute

$$\sigma_x = s \cdot x + \text{PRF}'(\text{key}', \text{id}) \text{ over } \mathcal{R}_p.$$

$\{\mathbf{k}_C^{(t)}\} \leftarrow \text{EvalKey}(\text{sk}, C^{\times T}, \{\mathbf{id}^{(t)}\})$: Read key' from sk , and for every input wire i , compute the keys for all SIMD instances and pack them. View the packed key as the 0-share of an authenticated share: (Denote $\mathbf{x}^{(i)} := (\mathbf{x}^{(0)}[i], \dots, \mathbf{x}^{(T-1)}[i])$ the SIMD values on i -th wire.)

$$\begin{aligned} k^{(i,t)} &\leftarrow \text{PRF}'(\text{key}', \mathbf{id}^{(t)}[i]), \\ \langle s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A \rangle_0 &\leftarrow \sum_{t \in [T]} k^{(i,t)} \cdot X^t \text{ over } \mathcal{R}. \end{aligned}$$

Next process every gate g in C , with input wires i, j and output wire k in topological order. We maintain the invariant that each SIMD (intermediate) wire corresponds to an authenticated share of a Type-A SIMD encoding.

- Read key, pd from evk , and prepare $\text{pd}^{(k)}$ by replacing the seed in pd (See Construction 9) with $\text{seed}^{(k)} \leftarrow \text{PRF}(\text{key}, k)$.
- For addition gate, compute

$$\langle s \cdot \mathcal{E}_{\mathbf{x}^{(k)}}^A \rangle_0 \leftarrow \langle s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A \rangle_0 + \langle s \cdot \mathcal{E}_{\mathbf{x}^{(j)}}^A \rangle_0 \text{ over } \mathcal{R}. \quad (14)$$

- For multiplication gate, compute

$$\langle s \cdot \mathcal{E}_{\mathbf{x}^{(k)}}^A \rangle_0 \leftarrow \text{SIMDMultKey}(\text{pd}^{(k)}, \langle s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A \rangle_0, \langle s \cdot \mathcal{E}_{\mathbf{x}^{(j)}}^A \rangle_0). \quad (15)$$

Finally, for the i -th output wire, unpack the corresponding authenticated shares of SIMD encodings $\langle s \cdot \mathbf{z}^{(i)} \rangle$, and key-switch to the desired secrets $\mathbf{s}^{*,(t)}$ chosen during KeyGen.

$$\begin{aligned} \{ \langle s \cdot \mathbf{z}^{(i,t)} \rangle \}_{0,t \in [T]} &\leftarrow \text{UP.UnPack}(\text{UP.evk}, \langle s \cdot \mathbf{z}^{(i)} \rangle_0), \\ \langle \text{Bits}(\mathbf{s}^{*,(t)}[i])[j] \cdot \mathbf{z}^{(i,t)} \rangle_0 &\leftarrow \text{KS.KeySW}_0(\text{KS.evk}^{(i,j,t)}, \langle s \cdot \mathbf{z}^{(i,t)} \rangle_0), \\ \mathbf{k}_C^{(t)}[i] &:= \langle \mathbf{s}^{*,(t)}[i] \cdot \mathbf{z}^{(i,t)} \rangle_0 \leftarrow \sum_{j \in [\lambda]} 2^j \cdot \langle \text{Bits}(\mathbf{s}^{*,(t)}[i])[j] \cdot \mathbf{z}^{(i,t)} \rangle_0. \end{aligned}$$

Output $\{\mathbf{k}^{(t)}\}$.

$\{\sigma_{\mathbf{z}}^{(t)}\} \leftarrow \text{EvalTag}(\text{evk}, C^{\times T}, \{\mathbf{x}^{(t)}\}, \{\sigma_{\mathbf{x}}^{(t)}\})$: For every input wire i , pack the inputs and tags across SIMD instances, and view the packed tag as the 1-share of an authenticated share: (We abuse notations to denote $\mathbf{x}^{(i)} := (\mathbf{x}^{(0)}[i], \dots, \mathbf{x}^{(T-1)}[i])$ the SIMD values on i -th wire.)

$$\langle s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A \rangle_1 \leftarrow \sum_{t \in [T]} \sigma_{\mathbf{x}}^{(t)}[i] \cdot X^t \text{ over } \mathcal{R}.$$

Then, proceed analogously to EvalKey to derive $\sigma_{\mathbf{z}}^{(t)}[i] := \langle \mathbf{s}^{*,(t)}[i] \cdot \mathbf{z}^{(i,t)} \rangle_1$ for $i \in [\ell_z]$, $t \in [T]$, and output $\{\sigma_{\mathbf{z}}^{(t)}\}$.

Correctness: Let $\{\mathbf{x}^{(t)}\}_{t \in [T]}$ be an admissible input to a SIMD arithmetic circuit $C^{\times T}$. We write $\mathbf{x}^{(i)} = (\dots, x^{(i,t)}, \dots)_{t \in [T]}$, where $x^{(i,t)} := \mathbf{x}^{(t)}[i]$, to denote the SIMD values on the i -th input wire. We argue the invariant that for every intermediate wire i in C , EvalKey and EvalTag respectively derive the 0-th and 1-th share of an authenticated share $\langle s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A \rangle$, where $\mathbf{x}^{(i)}$ are the correct SIMD values on wire i , and s is the secret sampled during KeyGen.

- For an input wire i , the EvalKey, EvalTag algorithms first derive $k^{(i,t)}$ and $\sigma_{\mathbf{x}}^{(t)}[i]$ from their inputs. By construction, they form additive shares of $s \cdot x^{(i,t)}$ over $\mathcal{R}_p$:

$$\begin{aligned} \sigma_{\mathbf{x}}^{(i)}[i] &= s \cdot x^{(i,t)} + k^{(i,t)} \text{ over } \mathcal{R}_p \\ \Rightarrow \sum_t \sigma_{\mathbf{x}}^{(i)}[i] \cdot X^t &= \sum_t (s \cdot x^{(i,t)} + k^{(i,t)}) \cdot X^t = s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A + \sum_t k^{(i,t)} \cdot X^t \text{ over } \mathcal{R}_p \end{aligned}$$

This means the initial packed keys and tags derived in EvalKey, EvalTag form additive shares of $s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A$ over $\mathcal{R}_p$. By the setting of $p = B \cdot \lambda^{\omega(1)}$, we have

$$\|s \cdot \mathcal{E}_{\mathbf{x}^{(i)}}^A\|_{\infty} < B \cdot T \cdot \text{poly}(\lambda) \ll p.$$

Hence the additive shares over $\mathcal{R}_p$ also form additive shares over $\mathcal{R}$ with overwhelming probability over the (pseudo-)randomness of $k^{(i,t)}$.

- For an output wire k of addition gate (see Equation 14), the invariant holds trivially.
- For an output wire k of multiplication gate (see Equation 15), the invariant holds with overwhelming probability over the (pseudo-)randomness of $\text{pd}^{(k)}$, by Lemma 6.

Finally, by the correctness of `UnPack` and `KeySW`, the output shares by `EvalKey` and `EvalTag` satisfy the desired algebraic format with overwhelming probability.

Security: We prove security of Construction 11 and hence Theorem 7.

Theorem 7. *Construction 11 is a SIMD aHMAC scheme for bounded integer arithmetic circuits, with secret domain $\mathcal{S} = [2^\lambda]$, per Definition 10, assuming circular-secure PRLWE with respect to $(\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}})$ as specified in Figure 1 and 2.*

Proof of Theorem 7. We have verified correctness above, and now focus on proving security. For any SIMD factor $T(\lambda) < \text{poly}(\lambda)$ input bound $B(\lambda) < 2^{\text{poly}(\lambda)}$, and a secret domain $\mathcal{S} = [2^\lambda]$, we first describe the simulators $\text{Sim}_1, \text{Sim}_2$ required by security.

- $\text{Sim}_1(1^{\ell_z})$ simulates $\text{evk} = (\widetilde{\text{key}}, \widetilde{\text{pd}}, \widetilde{\text{UP.evk}}, \{\widetilde{\text{KS.evk}}^{(i,j,t)}\})$ as follows.
 - Determine public parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ as in `KeyGen`.
 - Simulate $\widetilde{\text{key}} \leftarrow \{0, 1\}^\lambda$ at random.
 - Simulate $\widetilde{\text{pd}} := (\widetilde{\text{CC.evk}}, \widetilde{\text{CA.evk}}, \widetilde{\text{AB.evk}}, \widetilde{\text{aHMAC.pd}})$ as follows:

$$\begin{aligned}
\widetilde{\text{CC.evk}} &:= \{a^{(i)}, b^{(i)}\}_{i \in [\lceil \log T \rceil]} \text{ for } a^{(i)}, b^{(i)} \leftarrow \mathcal{R}_q, \\
\widetilde{\text{CA.evk}} &:= (a', b') \text{ for } a', b' \leftarrow \mathcal{R}_q, \\
\widetilde{\text{AB.evk}} &:= (a'', b'') \text{ for } a'', b'' \leftarrow \mathcal{R}_q, \\
\widetilde{\text{aHMAC.pd}} &:= (\text{pp}, \text{seed}, a, b, c), \text{ for } \text{seed} \leftarrow \{0, 1\}^\lambda, a, b, c \leftarrow \mathcal{R}_q,
\end{aligned} \tag{16}$$

- Simulate $\widetilde{\text{UP.evk}}$ as follows

$$\widetilde{\text{UP.evk}} := \{a'^{(i)}, b'^{(i)}\}_{i \in [\lceil \log T \rceil]}, \text{ where } a'^{(i)}, b'^{(i)} \leftarrow \mathcal{R}_q.$$

- Simulate $\{\widetilde{\text{KS.evk}}^{(i,j,t)}\}$ as follows

$$\widetilde{\text{KS.evk}}^{(i,j,t)} := (a''', b''') \text{ where } a''', b''' \leftarrow \mathcal{R}_q.$$

- $\text{Sim}_2()$ simulates authentication tags for (distinct) ids as $\tilde{\sigma} \leftarrow \mathcal{R}_p$.

We show a series of hybrid experiments that transitions from $\text{Hyb}_0 = \text{Exp}_{\text{priv}}^{\mathcal{A},0}(\lambda)$ to $\text{Hyb}_3 = \text{Exp}_{\text{priv}}^{\mathcal{A},1}(\lambda)$.

Hyb₀: The experiment $\text{Exp}_{\text{priv}}^{\mathcal{A},0}(\lambda)$ first receives a vectors $\mathbf{s}^{*,(t)} \in [2^\lambda]^{\ell_z}$ and computes a evaluation key $\text{evk} = (\text{key}, \text{pd}, \text{UP.evk}, \{\text{KS.evk}^{(i,j,t)}\})$ that's sent to the adversary. We briefly recall the components in evk :

- key is a random $(\lambda\text{-bit})$ string;
- $\text{pd} := (\text{CC.evk}, \text{CA.evk}, \text{AB.evk}, \text{aHMAC.pd})$, where
 - The evaluation keys $\text{CC.evk}, \text{CA.evk}, \text{AB.evk}$ consists of RLWE with circular security samples of the form $a, b = s \cdot a + e + \varphi(s) \cdot \Delta$, for a (global) secret s sampled from $\mathcal{D}_{\text{sk}}$ as specified in Section 4, and various coefficient mappings φ .

- The **aHMAC.pd** consists of a random seed and circular-secure PRLWE samples of the form $a, b = s \cdot a + e_1, c = s^2 + e_2 - s \cdot \Delta$.
- **UP.evk** consists of RLWE with circular security samples of the form $a, b = s \cdot a + e + \varphi(s) \cdot \Delta$, similarly to the other SIMD encoding conversion keys.
- **KS.evk**^(i,j,t) consists of RLWE ciphertexts of the form $a, b = s \cdot a + e + \text{Bits}(\mathbf{s}^{*,(t)}[i])[j] \cdot \Delta$.

Then, the experiment samples a secret PRF key', and for every received query $x \in [B]$, **id**, computes a tag $\sigma \leftarrow s \cdot x + \text{PRF}'(\text{key}', \text{id})$ over $\mathcal{R}_p$.

Hyb₂: This hybrid simulates all tags at random: $\sigma \leftarrow \mathcal{R}_q$. By the security of PRF', we have $\text{Hyb}_2 \approx_c \text{Hyb}_1$. Note that the global secret s is only used in the circular-secure PRLWE samples and ciphertexts as described above, and in particular no longer used for computing the tags.

Hyb₃: This hybrid simulates all circular-secure PRLWE samples and ciphertexts as random elements from $\mathcal{R}_q$, as described in the construction of **Sim₁**. By the circular-secure PRLWE assumption, we have $\text{Hyb}_3 \approx_c \text{Hyb}_2$. Note that **Hyb₃** is the as $\text{Exp}_{\text{HSS}}^{\mathcal{A},1}(\lambda)$.

By a standard hybrid argument, we conclude $\text{Hyb}_0 = \text{Exp}_{\text{priv}}^{\mathcal{A},0}(\lambda) \approx_c \text{Hyb}_3 = \text{Exp}_{\text{HSS}}^{\mathcal{A},1}(\lambda)$. $\square$

5.7 Concrete Efficiency

Now, we move to the concrete efficiency analysis of our SIMD constructions. **SIMD HSS**. We start with the concrete efficiency analysis of our SIMD HSS compared to directly using [BKS19]. We focus on the **Mult** and **SIMDMult** comparisons as this operation is the most time-consuming step for HSS evaluation.

We first count **HSS.Mult** in [BKS19]: A single multiplication takes a share of two ring elements (both in coefficient form) and a ciphertext of 4 ring elements (all in NTT form), to produce a new share of two ring elements (in coefficient form). Thus, the whole process takes 4 of NTT/INTT, 4 element-wise multiplications, 4 additions, and 2 roundings.

For our construction, the multiplication takes an additional process of **ClearCoeff** and **ConvCA**. Since **Mult** outputs both shares in NTT form, we can directly input them into **ClearCoeff**. In total, these two processes require $\log(n)/2 + 1$ key-switchings for **ClearCoeff** and one **ConvCA**. Therefore, our entire process takes $4 + 3 \cdot (\log(n)/2 + 1)$ of NTT/INTTs, $4 + 2 \cdot (\log(n)/2 + 1)$ element-wise multiplications, $4 + 2 \cdot (\log(n)/2 + 1)$ additions, and $2 + (\log(n)/2 + 1)$ roundings.

However, recall that our secrets require the second half to be zeros. Therefore, the ring dimension needed for **SIMDMult** is $2 \times$ the ring dimension needed for **Mult**. Overall, our construction has an overhead of $\lesssim \log(n) \times$ compared to [BKS19], while supporting a SIMD factor of $\sqrt{n/2}$, thus achieving a speed up of $\approx \sqrt{n/2}/\log(n)$ (in terms of amortized efficiency). The counts are summarized in Table 1. We discuss the concrete parameter selections together with some implementation-level optimizations in Section 7.

Now, we move to the concrete efficiency analysis of our SIMD aHMAC compared to directly using [ILL25a]. Mainly, we compare our **SIMDMultKey** and **SIMDMultTag** with **MultKey** and **MultTag** from [ILL25a] for the same reason.

SIMDMultTag versus MultTag In [ILL25a], a single multiplication of tag **MultTag** takes two secret shares $\langle x \rangle_1$ and $\langle y \rangle_1$ and their corresponding values x, y all in $\mathcal{R}$ (again, all in NTT form). These can also be seen as $\langle \mathbf{s} \cdot x \rangle_1$ and $\langle \mathbf{s} \cdot y \rangle_1$ with $\langle x \rangle_0, \langle y \rangle_0 = 0$ (i.e., a trivial share of x, y). It also takes public data $(a, b, c) \in \mathcal{R}_q^3$ (in NTT form) and outputs $\langle z \rangle$ and z (in NTT form).

Essentially, it computes $d \leftarrow -a \cdot \langle x \rangle_1 \cdot \langle y \rangle_1 + b \cdot (x \cdot \langle x \rangle_1 + y \cdot \langle y \rangle_1) - c \cdot x \cdot y$ followed by a rounding and rerandomization. In total, this process takes 6 NTT/INTT operations, 7 element-wise multiplications, 4 ring additions (1 from rerandomization), and 1 rounding.

For our **SIMDMultTag**, it takes an additional **ConvAB**, **ClearCoeff** and **ConvCA**, which together takes $\log(n)/2+2$ key-switchings. Thus, for us, in total, it takes $6+3 \cdot (\log(n)/2+2)$ NTT/INTTs, $7 + 2 \cdot (\log(n)/2 + 2)$ element-wise multiplications, $4 + 2 \cdot (\log(n)/2 + 2)$ ring additions, and $1 + \log(n)/2 + 2$ rounding operations. With a SIMD factor of $\sqrt{n/2}$, we achieve the amortized efficiency improvement of $\approx \sqrt{n/2}/\log(n)$.

SIMDMultKey for our SIMD aHMAC. On the other hand, **SIMDMultKey** is more lightweight. In [ILL25a], **MultKey** takes two shares in $\mathcal{R}$ as input (in NTT form), and a in pd to output one share in $\mathcal{R}$ (in NTT form). It can be achieved in 3 NTT/INTT, 2 element-wise multiplications, 1 addition, plus 1 rounding.

For our **SIMDMultKey**, it requires $\log(n)/2 + 2$ additional key-switchings. However, recall that $\langle v \rangle_0$ is zero, thus making the key-switching slightly cheaper, as analyzed in Section 4.4. In total, for our **SIMDMultKey**, it requires $3 + 2 \cdot (\log(n)/2 + 2)$ of NTT/INTT, $2 + \log(n)/2 + 2$ of element-wise multiplications, $1 + 2 \cdot (\log(n)/2 + 2)$ additions, and $1 + \log(n)/2 + 2$ of roundings. Hence, similarly, we have the amortized efficiency improvement of $\approx \sqrt{n/2}/\log(n)$.

Lazy coefficient clearance. Recall that in HSS and aHMAC, we need to compute **ClearCoeff** and **ConvCA** after calling the underlying HSS mult or aHMAC mult. However, this does *not* have to be done immediately. For example, if an addition is followed by two multiplications, the **ClearCoeff** and **ConvCA** can be computed *after* the additions are done. More formally, suppose the circuit needs to compute $x \cdot y + x' \cdot y'$, a regular (non-SIMD) multiplication can be used to first compute $v \leftarrow xy, v' \leftarrow x'y'$. Instead of computing $\text{ConvCA}(\text{ClearCoeff}(xy)) + \text{ConvCA}(\text{ClearCoeff}(x'y'))$, we can first compute $v'' \leftarrow v+v'$ and then compute $\text{ConvCA}(\text{ClearCoeff}(v''))$. This saves one set of **ClearCoeff** and **ConvCA**, which is the most expensive component of our constructions. This can be generalized to w multiplications, followed by $w - 1$ additions, for any $w > 1$. Let us call a circuit with such a special structure (i.e., on average, w multiplications are summed right after these multiplications) a w -wide circuit. For a w -circuit, our efficiency improvement for HSS and aHMAC, is instead $\approx \sqrt{n/2} \cdot (w - 1)/\log(n)$. Later in Section 6.5, we use this optimization to reduce the cost of garbling.

6 SIMD Garbling with Amortized 1-Bit per Gate

In this section, we follow the framework of [ILL25b] to combine our SIMD aHMAC (Construction 11) and SIMD HSS (Construction 8) evaluation procedures to obtain a SIMD Boolean garbling scheme with a 1-bit per gate amortized garbling size.

As an practical optimization, we consider a variant with a preprocessing phase (Definition 12) where the garbler sends a message to the evaluator, independent of the circuit C except for an upperbound on its size, and both parties can perform possibly heavy precomputations.

6.1 Garbling Definitions

Definition 11 (Boolean Garbling). *A Boolean garbling scheme for a class of Boolean circuit $\mathcal{C}$ consists of two efficient algorithms:*

- **Garb** $(1^\lambda, C \in \mathcal{C})$ takes a circuit $C : \{0, 1\}^{\ell_x} \rightarrow \{0, 1\}^{\ell_z}$, and outputs a garbling $\hat{C}$, and input key functions $\{K^{(i)}\}_{i \in [\ell_x]}$, where each key function $K^{(i)}$ maps an input bit $\mathbf{x}[i]$ to a label $L^{(i)} \in \{0, 1\}^\lambda$.

- $\text{Eval}(C, \widehat{C}, \{L^{(i)}\}_{i \in [\ell_x]})$ takes a circuit C , a garbling $\widehat{C}$, and input labels $L^{(i)}$ (corresponding to some input $\mathbf{x} \in \{0, 1\}^{\ell_x}$). It outputs the evaluation result $\mathbf{z} \in \{0, 1\}^{\ell_z}$.

Correctness: For every polynomial $p(\lambda)$, there exists a negligible function $\text{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, circuits $C \in \mathcal{C}$ with size $|C| \leq p(\lambda)$, and inputs $\mathbf{x} \in \{0, 1\}^{\ell_x}$, the following holds:

$$\Pr \left[\begin{array}{c} \text{Eval}(C, \widehat{C}, \{L^{(i)}\}) \\ = C(\mathbf{x}) \end{array} \middle| \begin{array}{c} (\widehat{C}, \{K^{(i)}\}) \leftarrow \text{Garb}(1^\lambda, C), \\ L^{(i)} = K^{(i)}(\mathbf{x}[i]). \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Security: There exists an efficient simulator Sim such that for every polynomial $p(\lambda)$ sequence of circuits $\{C_\lambda\}$ where $|C_\lambda| \leq p(\lambda)$, and sequence of inputs $\{\mathbf{x}_\lambda\}$, the following holds (suppressing the subscript λ for brevity):

$$\left\{ \text{Sim}(1^\lambda, C, C(\mathbf{x})) \right\}_\lambda \approx_c \left\{ \widehat{C}, \{L^{(i)}\}, \left| \begin{array}{c} (\widehat{C}, \{K^{(i)}\}) \leftarrow \text{Garb}(1^\lambda, C), \\ L^{(i)} = K^{(i)}(\mathbf{x}[i]). \end{array} \right. \right\}_\lambda$$

Definition 12 (SIMD Boolean Garbling with Preprocessing). A SIMD Boolean garbling scheme with preprocessing is a tuple of algorithms $(\text{GarbPre}, \text{GarbC}, \text{EvalPre}, \text{EvalC})$ with the following syntax

- $\text{GarbPre}(1^\lambda, 1^T, B_C)$ takes a SIMD factor T and an upperbound B_C on the size of C as inputs. It outputs a public data pd and a private state st .
- $\text{GarbC}(\text{st}, C^{\times T})$ takes a state st and a SIMD Boolean circuit $C^{\times T}$ as input, and outputs a garbled circuit $\widehat{C}$ and input key functions $\{K^{(i,t)}\}_{i \in [\ell_x], t \in [T]}$.
- $\text{EvalPre}(\text{pd})$ takes the public data pd as input, and outputs a state st' as output.
- $\text{GarbC}(\text{st}', \widehat{C}, \{L^{(i,t)}\}_{i \in [\ell_x], t \in [T]})$ takes the state st' , a garbled circuit $\widehat{C}$, and input labels $\{L^{(i,t)}\}$. It outputs the evaluation results $\{\mathbf{z}^{(t)}\}_{t \in [T]}$.

For every efficiently computable polynomial $T : \mathbb{Z} \rightarrow \mathbb{Z}$, we require the composed algorithms $(\text{Garb}^T, \text{Eval})$, respectively from $(\text{GarbPre}, \text{GarbC})$ and $(\text{EvalPre}, \text{EvalC})$, to be a garbling scheme for the class of T -SIMD Boolean circuits $\mathcal{C}^{\times T}$. The composition is as follows.

- $\text{Garb}^T(1^\lambda, C^{\times T})$: Run $(\text{st}, \text{pd}) \leftarrow \text{GarbPre}(1^\lambda, 1^T, |C|)$, and $(\widehat{C}, \{K^{(i,t)}\}) \leftarrow \text{GarbC}(\text{st}, C^{\times T})$. Output $\widehat{C}' = (\text{pd}, \widehat{C})$, and $\{K^{(i,t)}\}$.
- $\text{Eval}(\widehat{C}', \{L^{(i,t)}\})$: Read $\text{pd}, \widehat{C}$ from $\widehat{C}'$, and run $\text{st}' \leftarrow \text{EvalPre}(\text{pd})$. Then output $\{\mathbf{z}^{(t)}\} \leftarrow \text{EvalC}(\text{st}', \widehat{C}, \{L^{(i,t)}\})$.

6.2 The Overall Garbling Protocol

We follow the view of [ILL25b] to present our SIMD garbling scheme with preprocessing as a 2PC protocol (Figure 3 , 4) between a garbler P_G and an evaluator P_E with the following structure.

- In a Preprocess phase, both parties only knows a SIMD factor T and an upperbound B_C of the size of a SIMD circuit $C^{\times T}$. The garbler P_G sends public data pd to P_E , and both parties perform local computations to derive states st, st' respectively.
- In an Init phase, the garbler P_G knows SIMD inputs $\{\mathbf{x}^{(t)}\}$, and sends shares of the inputs to the evaluator P_E ;

- In an **Eval** phase, the two parties jointly evaluate gates of a SIMD Boolean circuit $C^{\times T} : \{0, 1\}^{\ell_x \times T} \rightarrow \{0, 1\}^{\ell_z \times T}$ in topological order;
- In a **Final** phase, the garbler P_G sends some decryption data to reveal the final output $\mathbf{z} \in \{0, 1\}^{\ell_z}$ to the evaluator P_E .

We note that all messages in this protocol are from the garbler P_G to the evaluator P_E . We further divide them into three parts, each satisfying a special property.

1. *Public data.* This is prepared by P_G during **Preprocess**. We require them to only depend on the SIMD factor T and an upperbound B_C of the circuit size.
2. *Input shares w.r.t. to the vector $\mathbf{x}$.* We require they are decomposable, i.e., each bit in this communication depends only on a single bit of $\mathbf{x}$.
3. *Garbling materials.* These include the decryption data during **Final**, and all communication during **Eval**. We require they are independent of the input $\mathbf{x}$.

Now, we can directly “compile” the above 2PC protocol into a SIMD garbling scheme with preprocessing as follows.

- The **GarbPre** algorithm outputs the public data $\mathbf{pd}$ prepared by the garbler P_G during **Preprocess** and its state $\mathbf{st}$.
- The **EvalPre** algorithm outputs the state $\mathbf{st}'$ of the evaluator P_E during **Preprocess**.
- The **GarbC** algorithm outputs (1) key functions $\{K_x^{(i)}\}$ such that labels $\{L_x^{(i)} := K_x^{(i)}(\mathbf{x}[i])\}$ exactly equal to the input shares w.r.t. $\mathbf{x}$, and (2) a garbled circuit $\hat{C}$ that contains the garbling materials.
- The **EvalC** algorithm performs all steps of P_E in the 2PC protocol (after **Preprocess**) to recover the final output.

6.3 The Gate Evaluation Sub-protocol

We describe our core sub-protocol **SIMDGateEval** for SIMD evaluation of an arbitrary Boolean gate (with 2 inputs and 1 output) in Figure 5.

At highlevel, our construction modifies the blueprint of [ILL25b] by leveraging our preprocessing phase to minimize the computation during (online) gate evaluations. In more detail, the construction proceeds in two steps:

1. Two parties start holding authenticated shares $\langle s\mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle$, $\langle s\mathcal{E}_{\bar{\mathbf{x}}^{(j)}}^A \rangle$ of the masked SIMD inputs $\bar{\mathbf{x}}^{(i)}$, $\bar{\mathbf{x}}^{(j)}$, and P_E also holds $\bar{\mathbf{x}}^{(i)}$, $\bar{\mathbf{x}}^{(j)}$. They derive authenticated shares (under $\mathbf{s} := (1, s)$) $\langle \mathbf{s} \cdot \mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle$ of the masked SIMD output through *local* computation.
2. The garbler P_G reveals its share $\langle \mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle_0$ to the evaluator P_E , who recovers $\bar{\mathbf{x}}^{(k)}$ in the clear.

Our modification applies to the first step, which we explain below.

- The parties first run SIMD aHMAC multiplications (and additions) to evaluate the indicator polynomial $p_{\alpha, \beta} : \{0, 1\}^2 \rightarrow \{0, 1\}$ for $\alpha, \beta \in \{0, 1\}$ over the input shares. The results are $\{\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}_{\alpha, \beta}}^A \rangle\}_{\alpha, \beta \in \{0, 1\}}$, where

$$z_{\alpha, \beta}^{(t)} := \mathbf{z}_{(\alpha, \beta)}[t] = 1 \iff (\alpha, \beta) = (\mathbf{x}^{(i)}[t], \mathbf{x}^{(j)}[t]). \quad (19)$$

Protocol SIMD2PC

The protocol runs between a garbler P_G and an evaluator P_E , to evaluate a SIMD Boolean circuit $C^{\times T} : \{0, 1\}^{\ell_x \times T} \rightarrow \{0, 1\}^{\ell_z \times T}$. It uses the following ingredients:

- SIMD HSS multiplication procedures SIMDPDGen , SIMDMult_b (Construction 7);
- SIMD aHMAC multiplication procedures SIMDPDGen , SIMDMultKey , SIMDMultTag (Construction 10);
- A suitable FHE scheme FHE supporting key-switching (Lemma 1);
- A PRF : $\{0, 1\}^\lambda \times [2^\lambda]^2 \rightarrow \{0, 1\}$.

Preprocess:

Inputs: P_G, P_E hold the SIMD factor T and a bound B_C on the circuit size of C .

Outputs: P_G, P_E hold states st, st' respectively.

1. P_G Let $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$, be public parameters specified in Figure 1 and 2.
2. P_G samples a PRF key $\text{key} \leftarrow \{0, 1\}^\lambda$ and prepares the public data pd as follows.

$$\begin{aligned}
 (\text{pk}, \text{FHE.evk}, s') &\leftarrow \text{FHE.KeyGen}(\text{pp}), \text{ct}_{s'}^{(i)} \leftarrow \text{FHE.Enc}(\text{pp}, \text{pk}, \text{key}[i]) \text{ for } i \in [\lambda], \\
 s &\leftarrow \mathcal{D}_{\text{sk}}, \text{ksk}' \leftarrow \text{KSKGen}'(\text{pp}, s', s), \\
 \text{pd}' &\leftarrow \text{HSS.SIMDPDGen}(\text{pp}, s), \text{pd}'' \leftarrow \text{aHMAC.SIMDPDGen}(\text{pp}, s), \\
 \text{pd} &:= (\text{pp}, \text{FHE.evk}, \text{ksk}', \{\text{ct}_i\}, \text{pd}', \text{pd}'')
 \end{aligned} \tag{17}$$

P_G sends pd to P_E .

3. Both P_G, P_E locally computes the following.
 - Run FHE.Eval on $\{\text{ct}_{s'}^{(i)}\}$ to derive encryptions $\{\text{ct}_{s'}^{(i,t)}\}$ of $\text{PRF}(\text{key}, i, t)$ for $i \in [B_C], t \in [T]$.
 - Key-switch and pack the evaluated ciphertexts:

$$\text{ct}_s^{(i,t)} \leftarrow \text{KeySW}'(\text{pp}, \text{ksk}', \text{ct}_{s'}^{(i,t)}), \text{ct}_s^{(i)} \leftarrow \sum_{t \in [T]} \text{ct}_{s'}^{(i,t)} \cdot X^{(T-1) \cdot t}.$$

P_E outputs $\text{st}' = (\text{pd}, \{\text{ct}_s^{(i)}\}_{i \in [B_C]})$; P_G outputs $\text{st} = (\text{st}', s, \text{key})$.

Fig. 3: The Preprocess phase of our SIMD 2PC protocol for Boolean circuits.

Protocol SIMD2PC Continued

Init: (Unchanged from [ILL25b].)

Inputs: P_G holds $\mathbf{st}$ and SIMD inputs $\{\mathbf{x}^{(t)}\}_{t \in [T]}$; P_E holds $\mathbf{st}'$.

Outputs: P_G, P_E jointly holds additive shares $\{\langle s\bar{\mathbf{x}}^{(t)} \rangle\}_{t \in [T]}$ (over $\mathcal{R}$); P_E also holds $\{\bar{\mathbf{x}}^{(t)}\}_{t \in [T]}$. P_G sends masked inputs $\bar{\mathbf{x}}^{(t)}$ and additive shares $\langle s\bar{\mathbf{x}}^{(t)} \rangle_1$ to P_E .^a

$$\begin{aligned}\bar{\mathbf{x}}^{(t)}[i] &\leftarrow \mathbf{x}[i] \oplus \text{PRF}(\text{key}, i, t), \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_0 \leftarrow \mathcal{R}_p, \\ \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_1 &\leftarrow s\bar{\mathbf{x}}^{(t)}[i] + \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_0 \text{ (over } \mathcal{R}), \text{ for } i \in [\ell_x], t \in [T].\end{aligned}$$

Eval:

Inputs: P_G, P_E both holds $C^{\times T}$, and jointly, $\{\langle s\bar{\mathbf{x}}^{(t)} \rangle\}$; P_G also holds $\mathbf{st}$; P_E also holds $\mathbf{st}'$ and $\{\bar{\mathbf{x}}^{(t)}\}$.
Outputs: P_E holds $\{\bar{\mathbf{z}}^{(t)}\}_{t \in [T]}$.

1. P_G, P_E locally pack the shares of inputs into shares of SIMD encodings.

$$\langle s\mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle \leftarrow \sum_{t \in [T]} \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle \cdot X^t \text{ for } i \in [\ell_x].$$

2. P_G, P_E evaluate gates $g \in C$, with input wires i, j and output wire k , in the topological order while maintaining the following invariant:
 - (a) P_G, P_E jointly hold $\langle s\mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle, \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(j)}}^A \rangle$, where $\mathbf{x}^{(i)}$ denotes the SIMD values on wire i , and

$$\bar{\mathbf{x}}^{(i)}[t] = \mathbf{x}^{(i)}[i] \oplus \text{PRF}(\text{sk}, i, t). \quad (18)$$

- (b) P_E holds the masked wire values $\bar{\mathbf{x}}^{(i)}, \bar{\mathbf{x}}^{(j)}$.

To evaluate the gate g , P_G, P_E jointly call the sub-protocol **SIMDGateEval**.

$$\begin{aligned}& (P_G : \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle_0), (P_E : \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle_1, \bar{\mathbf{x}}^{(k)}) \\ & \leftarrow \text{SIMDGateEval}^{C,g} \left((P_G : \mathbf{st}, \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle_0, \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(j)}}^A \rangle_0), \right. \\ & \quad \left. (P_E : \mathbf{st}', \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle_1, \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(j)}}^A \rangle_1, \bar{\mathbf{x}}^{(i)}, \bar{\mathbf{x}}^{(j)}) \right)\end{aligned}$$

Final: (Unchanged from [ILL25b].)

Inputs: P_G holds $\mathbf{st}$; P_E holds $\{\bar{\mathbf{z}}^{(t)}\}_{t \in [T]}$.

Outputs: P_E holds $\{\mathbf{z}^{(t)}\}_{t \in [T]}$.

P_G sends masks $\text{PRF}(\text{key}, o, t)$ on all output wires o to P_E , who recovers $\mathbf{z}^{(t)}$ by removing the masks.^b

^a $(\bar{\mathbf{x}}, \langle s\bar{\mathbf{x}} \rangle_1)$ jointly is what we call input shares in the overview text.

^b The Final phase can be avoided via an optimization: let **SIMDGateEval** compute z , instead of masked $\bar{z}$, for values on the output wires.

Fig. 4: The Init, Eval, Final phases of our SIMD 2PC protocol for Boolean circuits.

- Next, the parties run SIMD HSS multiplications (and additions) to evaluate the function $f_g : \{0, 1\}^7 \rightarrow \{0, 1\}$ defined as follows:

$$f_g(r^{(i,t)}, r^{(j,t)}, r^{(k,t)}, \{z_{\alpha,\beta}^{(t)}\}) := \sum_{\alpha,\beta} z_{\alpha,\beta}^{(t)} \cdot \underbrace{g(\alpha \oplus r^{(i,t)}, \beta \oplus r^{(j,t)}) \oplus r^{(k,t)}}_{f'_{g,\alpha,\beta}} \text{ over } \mathbb{Z}.$$

over SIMD ciphertexts $\text{ct}_s^{(i)}, \text{ct}_s^{(j)}, \text{ct}_s^{(k)}$, and authenticated shares $\{\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}^{(t)}}^A \rangle\}$ for $\alpha, \beta \in \{0, 1\}$.

The ciphertexts $\text{ct}_s^{(i)}$ are prepared in the preprocessing phase (using FHE) such that they encrypt Type-B SIMD encodings of the masks $\mathbf{r}^{(i)}$ on wire i . We note two facts about P_g :

- If $\mathbf{r}^{(i)}, \mathbf{r}^{(j)}, \mathbf{r}^{(k)}$, are the masks on wires i, j, k , then $\bar{\mathbf{x}}^{(k)} \leftarrow f_g^{(\times T)}(\mathbf{r}^{(i)}, \mathbf{r}^{(j)}, \mathbf{r}^{(k)}, \{\mathbf{z}_{\alpha,\beta}\})$ outputs the desired masked SIMD outputs $\bar{\mathbf{x}}^{(k)}$.
- $f_g^{(\times T)}$ can be written as a sum of SIMD monomials as follows: (over $\mathbb{Z}$)

$$f_g^{(\times T)}(\mathbf{r}^{(i)}, \mathbf{r}^{(j)}, \mathbf{r}^{(k)}, \{\mathbf{z}_{\alpha,\beta}\}) := \sum_{\alpha,\beta} \sum_{\gamma \in [8]} c_{\alpha,\beta,\gamma} \cdot \underbrace{\mathbf{z}_{\alpha,\beta} \odot f_{\gamma}^{(\times T)}(\mathbf{r}^{(i)}, \mathbf{r}^{(j)}, \mathbf{r}^{(k)})}_{m_{\alpha,\beta,\gamma}^{(t)}} \quad (20)$$

where $c_{\alpha,\beta,\gamma}$ are integer coefficients and $f_{\gamma}(r^{(i,t)}, r^{(j,t)}, r^{(k,t)})$ are monomials⁹ such that

$$f'_{g,\alpha,\beta} \equiv \sum_{\gamma} c_{\alpha,\beta,\gamma} \cdot f_{\gamma} \text{ over } \mathbb{Z}.$$

It follows from the above facts that the parties can derive authenticated shares $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{m}_{\alpha,\beta,\gamma}}^A \rangle$ by running iterated SIMD HSS multiplications between a share $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}^{(t)}}^A \rangle$ and the ciphertexts $\text{ct}_s^{(i)}, \text{ct}_s^{(j)}, \text{ct}_s^{(k)}$. Summing the shares of the monomials gives the desired share $\langle \mathbf{s} \cdot \mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle$.

6.4 Security Proof

We show the following theorem.

Theorem 8. *The SIMD garbling scheme with preprocessing compiled from the SIMD2PC protocol (Figure 3, and 4) satisfy Definition 12, assuming KDM-PRLWE with respect to $(\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}})$ specified in Figure 1 and 2.*

Proof of Theorem 8. The correctness of the SIMD garbling scheme follows from those of the SIMD aHMAC and SIMD HSS multiplication procedures. We focus on showing security. In particular, we construct a simulation simulator $\text{Sim}(1^\lambda, C, C^{\times T}(\{\mathbf{x}^{(t)}\}))$ as follows:

- Simulate input labels at random $L^{(i,t)} := (r \leftarrow \{0, 1\}, l \leftarrow \mathcal{R}_p)$ for $i \in [\ell_x], t \in [T]$.
- The garbled circuit $\hat{C}$ consists of three parts:
 - Public data $\text{pd} = (\text{FHE.evk}, \text{ksk}', \{\text{ct}_i\}_{i \in [\lambda]}, \text{pd}', \text{pd}'')$ as defined in Equation 17.
 - Strings $\mathbf{b}^{(i)} \in \{0, 1\}^T$ for all intermediate (including output) wires in C , corresponding to the communication from P_G in the sub-protocol SIMDGateEval (Figure 5).
 - Strings $\mathbf{r}^{(o)} \in \{0, 1\}^T$ for all output wires of C , corresponding to the communication from P_G in the Final phase (Figure 4).

⁹ There are at most $2^3 = 8$ of them, as the input variables are Boolean.

Sub-protocol SIMDGateEval^{C,g}

The sub-protocol runs between a garbler P_G and an evaluator P_E , to evaluate the gate g in a Boolean circuit C with SIMD inputs and outputs. Let the input wires be i, j and the output wire, k . It uses the same ingredients as the main protocol (Figure 3).

Inputs: P_G, P_E jointly hold shares $\langle s\mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle, \langle s\mathcal{E}_{\bar{\mathbf{x}}^{(j)}}^A \rangle$; P_G, P_E hold states $\mathbf{st}, \mathbf{st}'$ respectively.

Outputs: P_G, P_E jointly hold share $\langle s\mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle$; P_E also holds $\bar{\mathbf{x}}^{(k)}$.

1. P_G, P_E obtain additive shares $\langle s\bar{\mathbf{x}}^{(k)} \rangle$ and $\langle \bar{\mathbf{x}}^{(k)} \rangle$, through the following local computations.
 - Run SIMD aHMAC multiplications and additions to derive shares $\langle s \cdot \mathcal{E}_{\mathbf{z}_{\alpha,\beta}}^A \rangle$ as defined in Equation 19, for $\alpha, \beta \in \{0, 1\}$.
 - P_E derives $\mathbf{z}_{\alpha,\beta}$ in the clear. The parties locally define $\langle \mathcal{E}_{\mathbf{z}_{\alpha,\beta}}^A \rangle_0 \leftarrow 0$, and $\langle \mathcal{E}_{\mathbf{z}_{\alpha,\beta}}^A \rangle_1 \leftarrow \mathcal{E}_{\mathbf{z}_{\alpha,\beta}}^A$.
 - Run iterated SIMD HSS multiplications between the above shares $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{z}_{\alpha,\beta}}^A \rangle$, where $\mathbf{s} := (1, s)$, and the ciphertexts $\mathbf{ct}_s^{(i)}, \mathbf{ct}_s^{(j)}, \mathbf{ct}_s^{(k)}$ prepared in the Preprocess phase (and stored in $\mathbf{st}, \mathbf{st}'$). Derive shares of SIMD monomials $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{m}_{\alpha,\beta,\gamma}}^A \rangle$ for $\alpha, \beta \in \{0, 1\}$ and $\gamma \in [8]$ as defined in Equation 20. Then sum them to obtain $\langle \mathbf{s} \cdot \mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle \leftarrow \sum_{\alpha,\beta \in \{0,1\}, \gamma \in [8]} \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{m}_{\alpha,\beta,\gamma}}^A \rangle$.
2. P_G sends T bits $\mathbf{b} \in \{0, 1\}^T$, to P_E , who can then locally recover $\bar{\mathbf{x}}^{(k)}$:

$$\mathbf{b} \leftarrow \langle \mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle_0[:T] \bmod 2, \quad \bar{\mathbf{x}}^{(k)} \leftarrow \langle \mathcal{E}_{\bar{\mathbf{x}}^{(k)}}^A \rangle_1[:T] - \mathbf{b} \bmod 2.$$

Fig. 5: The Preprocess phase of our SIMD 2PC protocol for Boolean circuits.

Simulate the public data $(\text{FHE.evk}, \text{ksk}', \{\mathbf{ct}_{s'}^{(i)}\})$ related to FHE as follows. (See the KSKGen' specified in the proof of Lemma 1.)

$$\begin{aligned} (\text{FHE.pk}, \text{FHE.evk}, \text{FHE.sk}) &\leftarrow \text{FHE.KeyGen}(\text{pp}), \\ \mathbf{ct}_{s'}^{(i)} &\leftarrow \text{FHE.Enc}(\text{pp}, \text{FHE.pk}, 0), \text{ for } i \in [\lambda], \\ \text{ksk} &:= \{a_i, b_i \leftarrow \mathcal{R}_q\}_{i \in [\lceil \log q \rceil]}, \text{ evk}_2 := \{a'_i, b'_i\}_{i \in [\lceil \log q \rceil]}, \\ \text{pk}_2 &:= (a'', b'') \leftarrow \mathcal{R}_q^2, \text{ ct}_s := (a''', b''') \leftarrow \mathcal{R}_q^2, \\ \text{ksk}' &:= (\text{ksk}, \text{evk}_2, \text{pk}_2, \text{ct}_s). \end{aligned}$$

Simulate the public data pd', pd'' for SIMD HSS and aHMAC at random, as described in Equation 11 and 16 respectively. Then simulate the strings as $\mathbf{b}^{(i)} \leftarrow \{0, 1\}^T$, and program $\mathbf{r}^{(o)}$ such the evaluation over the simulated materials produces exactly the correct evaluation results $C^{\times T}(\{\mathbf{x}^{(t)}\})$.

Next, we show a series of hybrid experiments that transitions from generating the garbling materials $\widehat{C}, \{L^{(i,t)}\}$ honestly (Hyb_0) to simulating them as in Sim (Hyb_5).

Hyb₀: We briefly review the honest computation of $\{L^{(i,t)}\}$ and $\widehat{C} := (\text{pd}, \{\mathbf{b}^{(i)}\}, \{\mathbf{r}^{(o)}\})$:

- Sample a private PRF key $\leftarrow \{0, 1\}^\lambda$ at random, and a global secret $s \leftarrow \mathcal{D}_{\text{sk}}$. Compute the labels as follows:

$$\begin{aligned} \bar{\mathbf{x}}^{(t)}[i] &\leftarrow \mathbf{x}[i] \oplus \text{PRF}(\text{key}, i, t), \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_0 \leftarrow \mathcal{R}_p, \\ \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_1 &\leftarrow s\bar{\mathbf{x}}^{(t)}[i] + \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_0 \text{ (over } \mathcal{R}), \text{ for } i \in [\ell_x], t \in [T], \\ L^{(i,t)} &:= (\bar{\mathbf{x}}^{(t)}[i], \langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_1). \end{aligned} \tag{21}$$

- The public data $(\text{FHE.evk}, \text{ksk}', \{\text{ct}_{s'}^{(i)}\})$ in pd related to FHE are computed as follows:

$$\begin{aligned} (\text{pk}, \text{FHE.evk}, s') &\leftarrow \text{FHE.KeyGen}(\text{pp}), \\ \text{ct}_{s'}^{(i)} &\leftarrow \text{FHE.Enc}(\text{pp}, \text{pk}, \text{key}[i]) \text{ for } i \in [\lambda]. \\ \text{ksk}' &\leftarrow \text{KSKGen}'(\text{pp}, s', s), \end{aligned}$$

where $\text{ksk}' = (\text{ksk}, \text{evk}_2, \text{pk}_2, \text{ct}_{\text{sk}_2})$ are as follows: (See the KSKGen' in the proof of Lemma 1.)

- ksk consists of RLWE ciphertexts of the form $(a_i, b_i = s \cdot a_i + e_i + s' \cdot 2^i)$ for $i \in [\lceil \log q \rceil]$, with freshly sampled public element $a_i \leftarrow \mathcal{R}_q$ and noises $e_i \leftarrow \mathcal{D}_{\text{err}}$. (All RLWE / RLWE with circular security / KDM-PRLWE samples described below also have fresh public elements and noises.)
- evk_2 consists of RLWE with circular security samples of the form $(a_i, b_i = s \cdot a_i + e_i + s^2 \cdot 2^i)$ for $i \in [\lceil \log q \rceil]$.
- pk_2 consists of a RLWE sample of the form $(a, b = s \cdot a + e)$.
- ct_s consists of a RLWE with circular security sample of the form $(a, b = s \cdot a + e + s \cdot \Delta)$.
- Public data $\text{pd}' = (\text{CC.evk}, \text{CA.evk}, \text{HSS.pd})$ in pd for SIMD HSS are as follows:
 - The evaluation keys CC.evk , CA.evk consists of RLWE with circular security samples of the form $a, b = s \cdot a + e + \varphi(s) \cdot \Delta$, for various coefficient mappings φ .
 - HSS.pd consists of a random seed and public parameters pp .
- Public data $\text{pd}'' = (\text{CC.evk}, \text{CA.evk}', \text{AB.evk}, \text{aHMAC.pd})$ in pd for SIMD aHMAC are as
 - The evaluation keys CC.evk , $\text{CA.evk}'$, AB.evk consists of RLWE with circular security samples of the form $a, b = s \cdot a + e + \varphi(s) \cdot \Delta$, for various coefficient mappings φ .
 - The aHMAC.pd consists of a random seed and KDM-PRLWE samples of the form $a, b = s \cdot a + e_1, c = s^2 + e_2 - s \cdot \Delta$.
- The strings $\mathbf{b}^{(i)}$ for every intermediate wires, (including output wires) are computed by deriving garbler's (P_G) share $\langle \mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle_0$ according to the garbler's steps, and set

$$\mathbf{b}^{(i)} \leftarrow \langle \mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle_0[:T] \bmod 2.$$

- The strings $\{\mathbf{r}^{(o)}\}$ for every output wire o are computed as

$$\mathbf{r}^{(o)}[t] \leftarrow \text{PRF}(\text{key}, o, t) \text{ for } t \in [T].$$

Hyb₁: This hybrid computes the strings $\{\mathbf{b}^{(i)}\}$ and $\{\mathbf{r}^{(o)}\}$ differently.

- For every wire intermediate i , derive the evaluator's (P_E) share $\langle \mathcal{E}_{\bar{\mathbf{x}}}^A \rangle_1$ according to the evaluator's steps, and set

$$\begin{aligned} \mathbf{b}^{(i)} &\leftarrow \langle \mathcal{E}_{\bar{\mathbf{x}}^{(i)}}^A \rangle_1[:T] - \bar{\mathbf{x}}^{(i)} \bmod 2, \text{ where} \\ \bar{\mathbf{x}}^{(i)} &\leftarrow \mathbf{x}^{(i)}[t] \leftarrow \text{PRF}(\text{key}, i, t) \text{ for } t \in [T]. \end{aligned}$$

- For every output wire o , set

$$\mathbf{r}^{(o)}[t] \leftarrow \bar{\mathbf{x}}^{(o)}[t] - \mathbf{z}^{(o)}[t],$$

where $\mathbf{z}^{(o)}[t]$ are the correct evaluation results of the SIMD circuit.

By the correctness of our garbling protocol, we have $\text{Hyb}_1 \approx \text{Hyb}_0$.

Hyb₂: This hybrid computes the evaluator's (P_E) shares of the inputs at random $\langle s\bar{\mathbf{x}}^{(t)}[i] \rangle_1 \leftarrow \mathcal{R}_p$ instead of following Equation 21. This way of deriving the evaluator's share is statistically close to following Equation 21. In particular, the statistical distance is bounded by $n \cdot \|s\bar{\mathbf{x}}^{(t)}[i]\|_\infty/p$, which is negligible in our setting. Therefore, we have $\text{Hyb}_2 \approx \text{Hyb}_1$.

We note that the global secret s is now only used for generating the various RLWE, RLWE with circular security, KDM-PRLWE samples and ciphertexts as described in Hyb_0 . In particular, it's not used for deriving the evaluator's shares anymore.

Hyb₃: This hybrid samples all RLWE, RLWE with circular security, KDM-PRLWE samples and ciphertexts at random from $\mathcal{R}_q$. By the KDM-PRLWE assumption, we have $\text{Hyb}_3 \approx_c \text{Hyb}_2$. We note that the FHE secret key s' is no longer used anywhere in this hybrid, in particular not for generating ksk' anymore. (ksk' now consists of random elements in $\mathcal{R}_q$ instead of RLWE / RLWE with circular security samples.)

Hyb₄: This hybrid samples the FHE ciphertexts $\{\text{ct}_{s'}^{(i)}\}$ in pd as encryptions of 0, instead of bits of the PRF key:

$$\text{ct}_{s'}^{(i)} \leftarrow \text{FHE.Enc}(\text{pp}, \text{FHE.pk}, 0), \text{ for } i \in [\lambda].$$

By the semantic security of FHE, we have $\text{Hyb}_4 \approx_c \text{Hyb}_3$.

We note that the PRF key is only used for PRF evaluations (which produces masks on the wire values) in this hybrid, and in particular not used for generating the FHE ciphertexts anymore.

Hyb₅: This hybrid sets all masked wire values to be random: $\bar{\mathbf{x}}^{(i)} \leftarrow \{0, 1\}^T$ for all intermediate wire i . By the security of PRF, we have $\text{Hyb}_5 \approx_c \text{Hyb}_4$.

We note that Hyb_5 exactly corresponds to simulating the labels $\{L^{(i,t)}\}$ and garbled circuit $\hat{C}$ according to Sim .

By a standard hybrid argument, we conclude $\text{Hyb}_0 \approx_c \text{Hyb}_5$, which concludes the proof. $\square$

6.5 Concrete Efficiency Analysis

Lastly, we discuss the concrete efficiency. To this end, we first introduce an additional optimization. Currently, each gate takes one bit of communication at the cost of having 1 aHMAC multiplication plus 8 HSS multiplication calls. However, instead, we can use FHE to precompute the masks, which further reduces the number of HSS calls. This comes with a cost: the masks of input x, y are only known during online time. Thus, we cannot directly preprocess the correct mask, but instead precompute a totally different mask and substitute during the online time. This costs an extra bit per gate.

With this optimization, to perform a gate operation, with input $\langle s \cdot \mathcal{E}_x^A \rangle_b$ and $\langle s \cdot \mathcal{E}_y^A \rangle_b$ (both in NTT form). In total, we need 1 aHMAC multiplication call, and 5 HSS multiplication calls. It also requires aHMAC and HSS addition calls, but they are almost for free compared to the multiplications, and thus for readability, we do not count them. Overall, we similarly achieve $\sqrt{N/2}/\log(N)$ of efficiency improvement. However, note that this can be further improved using the idea below.

One example of such a circuit is our Garbling method. In our garbling gate operation, 3 of the 5 HSS multiplications are simply standard HSS multiplications. After the multiplication results are added, we can perform ClearCoeff and ConvCA . Thus, in total, one SIMD garbling gate evaluation for P_E takes $41 + 12 \cdot \log(N)$ of NTT/INTTs, $37 + 8 \cdot \log(N)$ element-wise multiplications, $34 + 8 \cdot \log(N)$ additions, and $16 + 4 \cdot \log(N)$ roundings. In comparison, one regular garbling gate evaluation (without SIMD) requires 26 of NTT/INTTs, 27 element-wise

multiplications, 24 additions, and 11 roundings. For P_G , it takes slightly fewer operations, since aHMAC for P_G is slightly cheaper. However, since the difference is little, we omit the difference.

7 Evaluation

Now, we evaluate the runtime of our constructions with concrete parameters.

7.1 Comparisons with non-SIMD HSS, aHMAC, and Garbling

We start by discussing how we choose parameters and then analyze the concrete runtime of each operation according to the analysis in Sections 4.4 and 5.7.

Trade-off between secret entropy and SIMD factor T . Before doing parameter analysis, we first introduce an additional optimization for concrete parameter selection. Recall that T needs to be a power of two to allow coefficient clearance. Furthermore, recall that as in Theorem 5, we require $T^2 + D \leq n$ where $D \leq n$ is the number of coefficients where the secret s is non-zero (since $\deg(s') \leq n - T^2 + T$ and $\deg(s) \leq D - T$)¹⁰. In prior sections, we use $T = \sqrt{n/2}$ to maximize efficiency. However, in practice, this may not be achievable. Specifically, for RLWE, n is usually a power-of-two while $\sqrt{n/2}$ might not be. Thus, in this case, we choose a smaller T but at the same time can have larger D such that the secret has higher entropy, providing better security according to LWE-estimator [APS15].¹¹

Parameter analysis. To choose parameters, we need to first choose the minimum ciphertext modulus q needed to guarantee the correctness of our constructions: i.e., each independent instance (among T SIMD instances) of circuit C evaluation is correct with probability $1 - 2^{-\kappa}$ for statistical security parameter κ . Then, we choose the smallest ring dimension n to guarantee sufficient computational security.

Let B be the input norm bound, σ be the error standard deviation of the RLWE ciphertexts (with error distribution $\mathcal{D}_{\text{err}}$ being Gaussian error with mean 0 and standard deviation σ). We start with aHMAC parameter setting. To this end, we start with the parameter analysis for the Auth operation (which is needed for every input) followed by the parameter analysis for SIMDMultTag (which imposes the strongest requirements during circuit evaluation).

1. To start, in Auth, we need to determine the modulus p such that $|s \cdot \mathcal{E}_x^A + k| \leq p$ (except for negligible probability), where $k \leftarrow_{\$} \mathcal{R}_p$. We know that $B_1 = |s \cdot \mathcal{E}_x^A| \leq T \cdot B \cdot |s|$. Then, for a circuit C with $|C|$ gates, each ring element with n coefficients, and $\log(n)$ rerandomizations¹², we need to apply the union bound to make sure that $|s \cdot \mathcal{E}_x^A + k| \leq p$ for each gate. Thus, we set $p \geq B_1 \cdot 2^\kappa \cdot |C| \cdot n \cdot \log(n)$.
2. Then, inside SIMDMultTag, recall that we need to call MultTag in [ILL24]: with input $\langle s \cdot \mathcal{E}_x^A \rangle_1 \cdot \langle s \cdot \mathcal{E}_y^B \rangle_1, \mathcal{E}_x^A, \mathcal{E}_y^B$, it computes $-a \cdot \langle s \cdot \mathcal{E}_x^A \rangle_1 \cdot \langle s \cdot \mathcal{E}_y^B \rangle_1 + b \cdot (\mathcal{E}_x^A \cdot \langle s \cdot \mathcal{E}_x^A \rangle_1 + \mathcal{E}_y^B \cdot \langle s \cdot \mathcal{E}_y^B \rangle_1) - c \cdot \mathcal{E}_x^A \cdot \mathcal{E}_y^B$.

¹⁰ In the theorem, we require $\deg(s') \leq n/2 + S$, and this is because we let $S \cdot T = T^2 = n/2$ to allow $T = \sqrt{n/2}$ to maximize the SIMD factor for efficiency. However, as we are about to explain, this can be tuned in a more fine-grained way.

¹¹ Note secrets with the last $n - D$ coefficients being 0 and the first D coefficients being regular secrets are widely used in FHE works Section 3.2, so similar to these works, we assume that this provides the same security as a ternary secret used sampled from the ring with dimension D .

¹² Recall that after each rounding, we need to perform one rerandomization. For each multiplication, we require $\log(n)/2 + c$ roundings, where c is a small constant. For better readability, we use $\log(n)$ directly, and this is reasonable since as shown in Table 1, $c \leq \log(n)/2$ for the n we choose below.

After this operation, we obtain $\langle \Delta \cdot v + e \rangle$ for some intended value v and error e . This error e has standard deviation $\sigma' = 2 \cdot p \cdot B \cdot T \cdot \sqrt{n} \cdot \sigma$. Error bound B_e is such that $1 - \text{erf}(B_e/(\sqrt{2} \cdot \sigma')) \leq 2^{-\kappa \cdot \log(|C|) \cdot \log(n) \cdot \log \log(n)}$ (where erf is the error function) [CGGI20]. To make sure that the rounding later can be done correctly, again, counting the circuit size $|C|$, number of coefficients n , and number of roundings per operation $\log(n)$, we need $\Delta \geq 2 \cdot B_e \cdot 2^\kappa \cdot |C| \cdot n \cdot \log(n)$.

3. Lastly, we just need to set the ciphertext modulus q . As mentioned in Requirement 2, we set $q = p \cdot \Delta$.

For HSS (Construction 8), the parameters are slightly different. In short, the error B_e is smaller: it is only $B \cdot \sqrt{n} \cdot B_\sigma$ (since inside HSS.Mult , we obtain $\langle \Delta \cdot x \cdot y + e_{\text{input}} \cdot y \rangle$). Thus, for q , we need $\log(p) + 1$ less bits.

For garbling (Fig. 5), recall that when switch between HSS and aHMAC for garbling, we only need to be in $\mathcal{R}$ in coefficient form, so even if they share different modulus, the switch between them can be easily done as long as they share the same T . Thus, we directly use the parameters for each primitive with $T = 2^6$.

Concrete parameters. For statistical security of 40 bits, $\kappa = 40$, We first set $B = |s| = 1$ (binary input, and $\mathcal{D}_{\text{sk}}$ generating a ring element where $D = n - T^2$ slots are uniform binary), $|C| = 2^{20}$ (as in [ACK23, GBK⁺24, ZWL⁺25]), and $\sigma = 3.2$ (for $\mathcal{D}_{\text{err}}$ being Gaussian error distribution with standard deviation being 3.2). With this, we set p to be 84 bits, B_e to be 97 bits, Δ to be 182 bits, and q to be 266 bits; then, $n = 2^{14}$, $T = 2^6$, $D = 3 \cdot 2^{12}$. This guarantees > 128 -bit computational security according to [APS15].

For HSS, we can use q being 175 bits (saving $\log(p) + 1$ bits). In this case, we can either set $n = 2^{13}$, $T = 2^5$, $D = 7 \cdot 2^{10}$, or $n = 2^{14}$, $T = 2^6$, $D = 3 \cdot 2^{12}$. These two parameter sets would yield similar amortized efficiency, but the former has slightly better total runtime. Thus, we provide runtime estimation for both, and use the latter for our garbling runtime estimation, to make sure that the HSS and aHMAC schemes share the same SIMD factor T .

Parameters for prior works. For prior aHAMC construction [ILL25a], parameters would be slightly different. Specifically, (1) they do not need to consider the SIMD factor T for norm growth (i.e., $B_1 = |B| \cdot |s|$ above), (2) they do not need to consider the $\log(n)$ rounding/rerandomization but instead only have 1 rounding/rerandomization, and (3) their secret key can be non-zero for all the n coefficients instead of only $n - T^2$. Therefore, for them, they can set q to be 236 bits. However, due to the inflexibility of RLWE, to guarantee > 128 -bit security, they also need $n = 2^{14}$.

For HSS [BKS19], similarly, q can be similarly reduced by 75 bits compared to prior aHMAC, and n can be reduced to 2^{13} .

Microbenchmarks. With these parameters, we can perform some microbenchmarks over the unit operations for us to estimate the overall runtime. Note that for our $q \approx 2^{266}$, we use $q = q_1 \cdot q_2 \cdots q_6$, where $q_1 \cdot q_2 = p$, each with ≈ 42 bits while q_3, q_4, q_5, q_6 each has ≈ 45 bits. This is so-called the Residue Number System (RNS), widely used in lattice-based cryptosystems to divide a huge number into smaller prime numbers using the Chinese Remainder Theorem. Each q_i is called an RNS limb. RNS ensures that the runtime of each $\mathcal{R}_q$ operation can be divided into several $\mathcal{R}_{q_i}$ operations, and each q_i can be < 64 bits within modern machine word size to guarantee satisfactory efficiency. Furthermore, for HEXL [BKS⁺21], NTT/INTT runtime is maximized with < 50 -bit primes, so instead of choosing q_3, q_4, q_5 each with ≈ 59 bits, we use q_3, q_4, q_5, q_6 . For HSS, we can use 4 RNS limbs (with q_3, q_4 being ≈ 46 bits).

	HSS		aHMAC				Garbling	
	Mult [BKS19]	SIMDMult Const. 10	MultKey [ILL25a]	SIMDMultKey Const. 7	MultTag [ILL25a]	SIMDMultTag Const. 7	GateEval [LWYY25, ILL25b]	SIMDGateEval Fig. 5
Total runtime (microsecond, μs)	392	1912	1755	8811	853	6013	3855	24357
Amortized efficiency (μs per input)	392	60	1755	138	853	94	3855	381
Amortized efficiency with GPU (μs per input)	67	7	96	8	49	6	439	30
Amortized efficiency with dedicated hardware (nanoseconds per input)	392	60	1755	138	853	94	3855	381

Table 2: Efficiency comparisons with prior works. See Section 7.1 for our parameter settings.

For prior aHMAC, similarly, 6 RNS limbs are necessary where $q_1 \cdot q_2 = p$ each with ≈ 37 bits and $q_3, \dots, q_6$ have ≈ 41 bits. For prior HSS, 4 RNS limbs are sufficient, where q_1, q_2 have ≈ 37 bits and q_3, q_4 have ≈ 44 bits.

Based on our benchmark on GCP C4-standard-2 Instance using SEAL library [SEA25] (with HEXL acceleration [BKS⁺21]), for $n = 2^{14}$, each main operation has the following runtime:

- NTT/INTT: 185 microseconds (6 RNS limbs), 129 microseconds (4 RNS limbs)
- Element-wise multiplication: 52 microseconds (6 RNS limbs), 38 microseconds (4 RNS limbs)
- Addition: 29 microseconds (6 RNS limbs), 17 microseconds (4 RNS limbs)
- Rounding: 165 microseconds (6 RNS limbs), 105 microseconds (4 RNS limbs)

Additionally, we benchmark for $n = 2^{13}$ as well.

- NTT/INTT: 58 microseconds (4 RNS limbs)
- Element-wise multiplication: 16 microseconds (4 RNS limbs)
- Addition: 7 microseconds (4 RNS limbs)
- Rounding: 48 microseconds (4 RNS limbs)

General performance. As shown in Table 2, in terms of amortized efficiency, our schemes are 6-13 times faster than prior works, since we supports SIMD over $2^6 = 64$ inputs.¹³ Note that these runtime remains the same for $B \leq 2^{10}$ since it does not change n nor the number of RNS limbs for $B \leq 2^{10}$ (and all the RNS limbs remain ≤ 50 bits for best HEXL performance). Additionally, for more specially structured circuits, we can achieve $\sim 50\times$ speedup, as discussed below.

7.2 Comparisons with SIMD HSS

As mentioned, [BKS19] supports naive SIMD of size n for ring dimension n . However, it requires an intermediate bound $\beta > n^d$ for branching program depth d , even for Boolean inputs, and an RLWE modulus being at least $q \geq n^{2d}$. This means that for a Boolean circuit of depth d_c , after compiled into an RMS program required by HSS, the modulus needs to be at least $n^{2^{d_c+1}}$. We

¹³ Our estimation is consistent with the implementation evaluation reported in [ZLY⁺26]. For example, their “BitGC-fast Eval” procedure effectively performs three HSS multiplications, taking a total of 1.66 ms, i.e., approximately ~ 0.55 ms per HSS multiplication. Our estimated cost is about ~ 0.4 ms per HSS multiplication, which is very close to their reported performance. The remaining difference mainly stems from the choice of ciphertext modulus q : their implementation uses a 240-bit modulus, whereas ours uses a 161-bit modulus, leading to an expected slowdown of roughly $1.5\times$ in their setting.

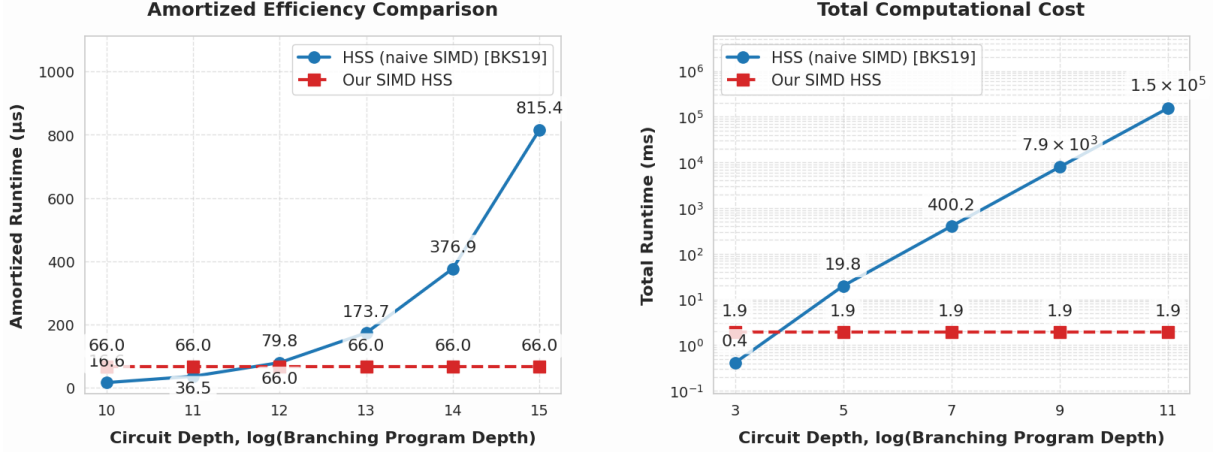


Fig. 6: Comparison with the naive SIMD HSS in [BKS19].

use this to give a generous estimation on the runtime of [BKS19], and show it in Fig. 6, where the amortized efficiency is shown in log-linear scale and total runtime in log-log scale.

Observe that for a Boolean circuit of depth $d_c = 12$ or larger, our SIMD HSS outperforms the naive SIMD method in terms of amortized computational efficiency. This gap becomes $> 100\times$ for a depth of $d_c = 15$. Furthermore, even at a smaller depth, e.g. $d_c = 11$, the naive SIMD HSS achieves better amortized efficiency at the cost of a large latency of 150 seconds per operation (while ours remain 1.9ms). Such a high latency could be a concern in many applications, despite its slightly better throughput.

7.3 Additional Comparisons and Discussion

w -wide circuits. As discussed in Section 5.7, our SIMD HSS and aHMAC shines for w -wide circuits – circuits where on average, every w multiplications get summed. For example, an inner product between two vectors of length w : $\sum_{i \in [w]} \mathbf{x}[i] \cdot \mathbf{y}[i]$ is a w -wide circuit, or more generally, a matrix multiplication between a matrix of size $k \times w$ and a vector of size $w \times 1$ for any $k \geq 1$. Such circuits are widely studied in the context of secure computation since it is widely applicable [VZ25, BN25, BCH⁺25, jLHH⁺21, BCH⁺24, HS18].

For such circuits, we can perform the lazy coefficient clearance technique (see Section 5.7). Since for such circuits, it makes more sense to consider bounded integer operations, following prior works [BKS19, ACK23], we choose an input bound of 2^{128} and choose the rest of the parameters similarly as discussed above, except that now we have $B = 2^{128}$ instead of $B = 2$.

We show the amortized efficiency of our constructions compared to prior works for an inner product of size w in Fig. 7 (we only compute the cost for multiplications since additions are almost for free in comparison). As w increases, our HSS and aHMAC obtain better speedup comparing to prior non-SIMD schemes, since the cost of coefficient clearance is amortized across more multiplications. As shown, with a $w = 64$, our SIMD aHMAC is $\sim 50\times$ faster than plain aHMAC [ILL24] and our SIMD HSS is $\sim 25\times$ faster than plain HSS [BKS19].

With hardware acceleration. In Table 2, we additionally show the runtime with hardware acceleration. For hardware acceleration, we start by considering GPU. According to [ÖS23], one NTT/INTT is ~ 7.65 microseconds for $n = 2^{13}$ and 4 RNS limbs (i.e., 4 NTT/INTT in parallel),

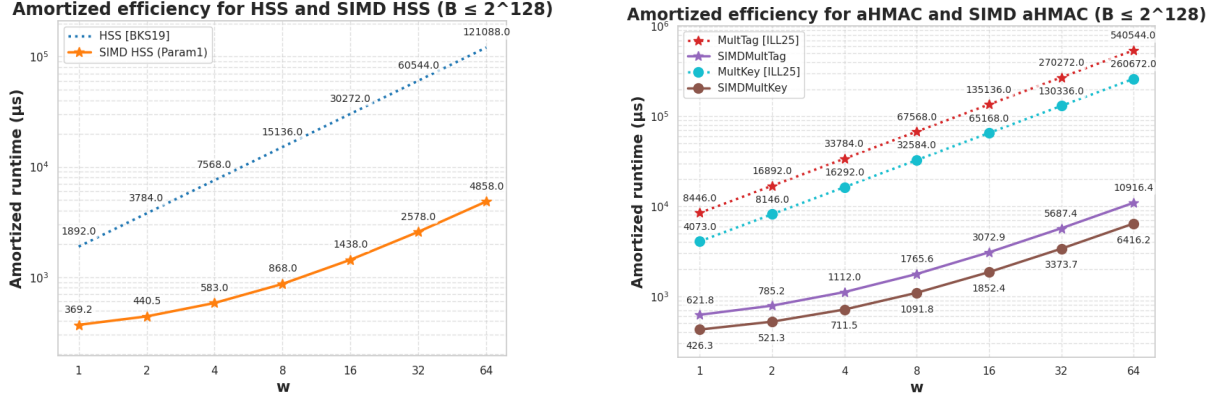


Fig. 7: Amortized efficiency as a function of parameter w (log-log scale). On the left, we compare HSS (dotted) and SIMD HSS (solid). On the right, we compare aHMAC (dotted) and SIMD aHMAC (solid).

~ 8 for $n = 2^{14}$ and 4 RNS limbs, ~ 8.5 for $n = 2^{14}$ and 6 RNS limbs¹⁴. Then, according to [ÖS24], element-wise multiplications and additions are both ~ 2 microseconds for $n = 2^{13}$ and ~ 2.65 microseconds for $n = 2^{14}$; rounding is ~ 11 microseconds for $n = 2^{13}$ and ~ 16 microseconds for $n = 2^{14}$. We assume these runtime remain similar for 4 and 6 limbs, which is reasonable as for the NTT/INTT, it has limited difference. With GPU, we can achieve an amortized efficiency on the order of several microseconds per multiplication.

Additionally, there exist works that use dedicated hardware to improve the efficiency of ring operations [SFK⁺21, JCT⁺25]. Based on [SFK⁺21], per NTT operation takes only ~ 45 nano seconds for $n = 2^{13}$ and ~ 179 nanoseconds for $n = 2^{14}$, which are both roughly $1000\times$ faster than a single CPU benchmark below. While unfortunately, in [SFK⁺21], there are no direct benchmarks on the speedup of the other unit operations, based on their overall tests [SFK⁺21, Tab. 4], NTT in general has a much smaller improvement over CPU compared to other operations. Similarly, in [JCT⁺25], the overall improvements over CPU is $> 1000\times$ for an FHE bootstrapping. Thus, for simplicity, we assume that such dedicated hardware is $1000\times$ faster for all the other unit operations, and estimate our runtime accordingly.

Relaxed HSS and aHMAC. In [ACK23], the authors introduced an alternative way to perform rounding and lifting to reduce the ciphertext modulus (from super-polynomial to polynomial) for HSS. However, it requires a change in reconstruction. At a high-level, the idea is to have a smaller modulus and a non-negligible rounding error probability. However, party 1 can see when the error is likely to happen. If it is in the range where the error is likely to happen, party 1 makes two separate roundings where one of them is correct. Then, party 0 is able to determine which one is correct and therefore recover using the correct share. If $m \geq 2$ errors happen, party 1 would keep a list of shares of 2^m size. Each operation following m errors has a computation overhead of 2^m as well (since all the shares need to be evaluated). In [ACK23], the authors provide a way to make sure $2^m = O(1)$ in expectation.

While this could improve the efficiency, this technique also imposes challenges to applications like secure 2-party computation, since the reconstruction algorithm should not have any privacy

¹⁴ Note that they do not have 6 NTT/INTT in parallel, so we estimate using their runtime for 4 and 16 NTT/INTTs in parallel.

requirement. Thus, we see this as a relaxed HSS instead of a full HSS. With this relaxed HSS, we are able to reduce the ciphertext modulus by ~ 70 bits (therefore only 2 RNS limbs required), therefore making n $2\times$ smaller as well. This allows our SIMD HSS to be roughly $4\times$ faster.

Similarly, this idea can be applied to aHMAC, achieving a similar improvement as well. However, due to the same issue, the resulting aHMAC no longer satisfies the requirement of aHMAC, thereby only a relaxed version of aHMAC. Due to this limitation, we do not discuss the concrete efficiency of such a relaxed version in detail.

Offline Costs. As discussed, our work primarily focuses on online efficiency. For completeness, we briefly address the efficiency of the offline garbling phase. As illustrated in Fig. 3 (in Section 2.4), the offline phase, both in our construction and in prior work, can be instantiated using any FHE scheme that supports a *key-switching* interface (cf. Appendix 3.3). This requirement is satisfied by BGV [BGV14] and BFV [Bra12, FV12], both of which support SIMD operations.

Indeed, a recent independent work [ZLY⁺26] presents a concretely efficient instantiation of the offline garbling phase. This work is fully complementary to ours: while it does not support SIMD during the online phase, thereby achieving online efficiency comparable to [LWYY25, ILL25b], its offline techniques are directly applicable to our construction. In particular, their *extract* procedure corresponds to the key-switching and packing step described in Fig. 3 (which is the dominating cost of their offline phase).

Based on their implementation results, the offline extraction cost is on the order of tens of milliseconds per gate. However, we expect our extraction to be $\tilde{O}(\sqrt{n})$ times faster. Specifically, their extraction involves (1) a homomorphic unpacking phase, which involves $2\sqrt{n}$ key-switchings, and (2) a coefficient-extracting phase, which transforms an RLWE ciphertext encrypting n coefficients into n RLWE ciphertexts each encrypting a single coefficient, taking $O(n)$ of key-switchings. The cost for (2) dominates. For us however, step (2) can be simplified: we only need to transform an RLWE ciphertext encrypting n coefficients into n/T RLWE ciphertexts each encrypting T coefficients, where $T = O(\sqrt{n})$, which can be achieved using only $\tilde{O}(n/T)$ key-switching. Therefore, we expect our extract procedure to be faster.

Acknowledgements

Jaehyung Kim was supported by the Stanford Graduate Fellowship. Hanjun Li and Huijia Lin were supported by a Simons Collaboration on the Theory of Algorithmic Fairness and an Amazon Research Award. Zeyu Liu was supported by Yale CADMY.

Bibliography

- [ACK23] Thomas Attema, Pedro Capitão, and Lisa Kohl. On homomorphic secret sharing from polynomial-modulus LWE. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 3–32. Springer, Cham, May 2023.
- [ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Cham, August 2022.
- [APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. <https://github.com/malb/lattice-estimator/tree/main>.
- [BAB⁺22] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Sponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. WAHC’22, 2022.
- [BCG⁺18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018.
- [BCG⁺19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.
- [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.
- [BCH⁺24] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. Plaintext-ciphertext matrix multiplication and FHE bootstrapping: Fast and fused. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 387–421. Springer, Cham, August 2024.
- [BCH⁺25] Fabrice Benhamouda, Caicai Chen, Shai Halevi, Yuval Ishai, Hugo Krawczyk, Tamer Mour, Tal Rabin, and Alon Rosen. Encrypted matrix-vector products from secret dual codes. CCS’25, 2025.
- [BCK⁺23] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 37–69. Springer, Cham, August 2023.
- [BCKS24] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 94–123. Springer, Cham, May 2024.

- [BD20] Zvika Brakerski and Nico Döttling. Lossiness and entropic hardness for ring-LWE. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 1–27. Springer, Cham, November 2020.
- [BGG⁺14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Berlin, Heidelberg, May 2014.
- [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.
- [BGI⁺18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.
- [BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3), July 2014.
- [BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.
- [BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Cham, May 2019.
- [BKS⁺21] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe D.M. de Souza, and Vinodh Gopal. Intel hexl: Accelerating homomorphic encryption with intel avx512-ifma52. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC ’21, page 57–62, New York, NY, USA, 2021. Association for Computing Machinery.
- [BN25] Mark Braverman and Stephen Newman. Practical secure delegated linear algebra with trapdoored matrices, 2025.
- [Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Berlin, Heidelberg, August 2012.
- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Berlin, Heidelberg, August 2011.

- [CDKS19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 395–412. ACM Press, November 2019.
- [CDKS21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 2021, Part I*, volume 12726 of *LNCS*, pages 460–479. Springer, Cham, June 2021.
- [CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.
- [CH18] Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 315–337. Springer, Cham, April / May 2018.
- [CHK⁺18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 360–384. Springer, Cham, April / May 2018.
- [CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017.
- [CKP25] Jung Hee Cheon, Minsik Kang, and Jai Hyun Park. Towards lightweight CKKS: On client cost efficiency. Cryptology ePrint Archive, Paper 2025/720, 2025.
- [Cle90] Richard Cleve. Towards optimal simulations of formulas by bounded-width programs. In *22nd ACM STOC*, pages 271–277. ACM Press, May 1990.
- [CM21] Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 842–870. Springer, Cham, October 2021.
- [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.
- [DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Berlin, Heidelberg, August 2016.
- [DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Berlin, Heidelberg, April 2015.
- [FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith, III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017*, volume 10592 of *LNCS*, pages 381–399. Springer, Cham, October 2017.

- [FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.
- [GBK⁺24] Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. Blind zkSNARKs for private proof delegation and verifiable computation over encrypted data. Cryptology ePrint Archive, Report 2024/1684, 2024.
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
- [GHPS12] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Ring switching in BGV-style homomorphic encryption. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 19–37. Springer, Berlin, Heidelberg, September 2012.
- [GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, Berlin, Heidelberg, April 2012.
- [GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, Berlin, Heidelberg, August 2012.
- [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.
- [GKP⁺13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.
- [GP25] Antonio Guimarães and Hilder V. L. Pereira. Fast amortized bootstrapping with small keys and polynomial noise overhead. CCS’25, 2025.
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.
- [HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, November 2023.
- [HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in HElib. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 93–120. Springer, Cham, August 2018.
- [ILL24] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct partial garbling from groups and applications. Cryptology ePrint Archive, Report 2024/2073, 2024.
- [ILL25a] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct Homomorphic MACs from Groups and Applications. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2505–2545, Los Alamitos, CA, USA, December 2025. IEEE Computer Society.
- [ILL25b] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 390–425. Springer, Cham, August 2025.

- [JCT⁺25] Siddharth Jayashankar, Edward Chen, Tom Tang, Wenting Zheng, and Dimitrios Skarlatos. Cinnamon: A framework for scale-out encrypted ai. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 133–150, New York, NY, USA, 2025. Association for Computing Machinery.
- [jLHH⁺21] Wen jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE Computer Society Press, May 2021.
- [KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015.
- [LKL⁺22] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.
- [LLKN23] Joon-Woo Lee, Eunsang Lee, Young-Sik Kim, and Jong-Seon No. Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 36–68, Singapore, 2023. Springer Nature Singapore.
- [LT22] Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 753–783. Springer, Cham, August 2022.
- [LTW24] Zeyu Liu, Eran Tromer, and Yunhao Wang. Group oblivious message retrieval. In *2024 IEEE Symposium on Security and Privacy*, pages 4367–4385. IEEE Computer Society Press, May 2024.
- [LW23] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{O}(1)$ polynomial multiplications. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 101–132. Springer, Singapore, December 2023.
- [LW24] Zeyu Liu and Yunhao Wang. Relaxed functional bootstrapping: A new perspective on BGV/BFV bootstrapping. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 208–240. Springer, Singapore, December 2024.
- [LWYY25] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: Garbled circuits with 1 bit per gate. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 437–466. Springer, Cham, May 2025.
- [MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 71–97. Springer, Cham, December 2024.
- [MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relin-earisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 426–458. Springer, Cham, August 2025.

- [NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Berlin, Heidelberg, August 2012.
- [ÖS23] Ali Şah Özcan and Erkey Savaş. Two algorithms for fast GPU implementation of NTT. Cryptology ePrint Archive, Report 2023/1410, 2023.
- [ÖS24] Ali Şah Özcan and Erkey Savaş. HEonGPU: a GPU-based fully homomorphic encryption library 1.0. Cryptology ePrint Archive, Report 2024/1543, 2024.
- [OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Cham, October 2021.
- [SEA25] Microsoft SEAL, 2025. <https://github.com/Microsoft/SEAL>.
- [SFK⁺21] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 238–252, New York, NY, USA, 2021. Association for Computing Machinery.
- [SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *DCC*, 71(1):57–81, 2014.
- [SW05] Amit Sahai and Brent Waters. Fuzzy identity based encryption. Eurocrypt, 2005.
- [VZ25] Vinod Vaikuntanathan and Or Zamir. Improving algorithmic efficiency using cryptography, 2025.
- [WYG⁺17] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: practical private queries on public data. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 299–313, USA, 2017. USENIX Association.
- [Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.
- [Zam22] Zama. Concrete: TFHE Compiler that converts python programs into FHE equivalent, 2022. <https://github.com/zama-ai/concrete>.
- [ZLY⁺26] Wenhao Zhang, Hanlin Liu, Kang Yang, Wenjie Lu, Yu Yu, Xiao Wang, and Chenkai Weng. BitGC made (more) efficient. Eurocrypt 2026 (To appear), 2026.
- [ZWL⁺25] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, Ben Fisch, and Xianhui Lu. Phalanx: An FHE-friendly SNARK for verifiable computation on encrypted data. CCS'25, 2025.

A SIMD Techniques Continued

A.1 Non-SIMD Operations

Packing and Unpacking. Although our SIMD arithmetic is complete in the sense that it supports (polynomially) unbounded number of vectorized operations, one may want to combine SIMD arithmetic with non-SIMD, individual arithmetic. In that case, we need conversions between SIMD and non-SIMD packing structures, which can be denoted as packing and unpacking. For non-SIMD packing structure, we mean by putting a bounded integer $x \in [B]$ into a constant term coefficient.

Suppose we have authenticated shares $\langle \mathbf{s} \cdot x_i \rangle$ for $0 \leq i < T$, where $x_i \in [B]$ is identified as an element in $\mathcal{R}$ for each i . To convert these authenticated shares into a single authenticated share with respect to the Type A encoding, we can compute

$$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{s}}^A \rangle = \sum_{i=0}^{T-1} \langle \mathbf{s} \cdot x_i \rangle \cdot X^i \in \mathcal{R}$$

where $\mathbf{x} = (x_0, x_1, \dots, x_{T-1}) \in [B]^T$. This automatically gives the packing algorithm.

For unpacking, we use homomorphic trace evaluation in an iterative manner.

Construction 12 (Unpacking). Let the parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be the same as in Construction 1.

$\text{evk} \leftarrow \text{UnpackKeyGen}(s, S)$: Input a parameter $T' \geq T$ which is a power-of-two for unpacking, the evaluation keys are generated by generating keys for the automorphisms $\varphi_i = X \mapsto X^{N/2^i+1}$ for each $0 \leq i < \log T'$. That is,

$$\text{UnpackKeyGen}(S) = (\text{KeyGen}(\varphi_i(s), s))_{0 \leq i < \log T'}$$

$(\langle \mathbf{s} \cdot x_i \rangle)_{0 \leq i < T} \leftarrow \text{Unpack}^A(\text{evk}, T', \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \rangle)$: Input an evaluation key evk generated via UnpackKeyGen , the unpacking parameter T' , and an authenticated share $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \rangle$, output an authenticated shares $(\langle \mathbf{s}' \cdot x_i \rangle)_{0 \leq i < T}$ where $\mathbf{x} = (x_0, x_1, \dots, x_{T-1}) \in [B]^T$. This is instantiate via the algorithm in Algorithm 4.

Algorithm 4 Unpack^A

Input: $u = \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \rangle$, $\text{evk}_i = \text{KeyGen}(\varphi_i(s), s)$ ($0 \leq i < \log T'$), where $\varphi_i = X \mapsto X^{N/2^i+1}$.

Output: $(\langle \mathbf{s} \cdot x_i \rangle)_{0 \leq i < T}$.

```

 $u_{0,0} \leftarrow u;$ 
for  $i \leftarrow 0$  to  $\log T' - 1$  do
  for  $j \leftarrow 0$  to  $2^i - 1$  do
     $u' \leftarrow \text{Aut}^{\varphi_i}(\text{evk}_i, u_{i,j});$ 
     $u_{i+1,2j} \leftarrow u_{i,j} + u';$ 
     $u_{i+1,2j+1} \leftarrow X^{-2^i} \cdot (u_{i,j} - u');$ 
  end for
end for
 $\mathbf{u} \leftarrow (u_{\log T', i})_{0 \leq i < T};$ 
 $\mathbf{u} \leftarrow \mathbf{u}/T';$ 
return  $\mathbf{u}$ 

```

Theorem 9 (Unpacking Correctness). *The unpacking algorithm according to Construction 12 is correct.*

Proof. Since key switching, addition, and scalar multiplication correctness ensures individual operations to be correct, it suffices to check that the resulting authenticated share vector has the desired values $\langle \mathbf{s} \cdot x_i \rangle$ for each $0 \leq i < T$.

To prove this, we claim that $u_{i,j}$ contains x_α 's with indices $\alpha \equiv_{2^i} j$, encoded as

$$2^i \cdot \sum_{k=0}^{T'/2^i-1} x_{2^i \cdot k + j} \cdot X^{2^i \cdot k}.$$

We prove this by mathematical induction on i . For $i = 0$, it is clear because $u_{0,0} = u$. If the property was true for i , then we prove for $i + 1$. For an integer $0 \leq j < 2^i$, we check both for $u_{i+1,2j}$ and $u_{i+1,2j+1}$. We observe that $u' \leftarrow \text{Aut}^{\varphi_i}(\text{evk}_i, u_{i,j})$ has

$$2^i \cdot \sum_{k=0}^{T'/2^i-1} (-1)^k \cdot x_{2^i \cdot k + j} \cdot X^{2^i \cdot k}$$

by definition of φ_i . Hence, we have that $u_{i+1,2j}$ and $u_{i+1,2j+1}$ have

$$2^{i+1} \cdot \sum_{k=0}^{T'/2^{i+1}-1} x_{2^{i+1} \cdot k + 2j} \cdot X^{2^{i+1} \cdot k} \text{ and } 2^{i+1} \cdot \sum_{k=0}^{T'/2^{i+1}-1} x_{2^{i+1} \cdot k + 2j+1} \cdot X^{2^{i+1} \cdot k},$$

respectively. By mathematical induction, we have that $u_{\log T', j}$ containing the desired elements x_j 's. This finishes the proof. $\square$

Of course, unpacking naturally give rotation and permutation over packed authenticated shares. We provide a more efficient instantiation of rotation.

Construction 13 (Rotation). Let the parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{err}}, \mathcal{D}_{\text{sk}}, B)$ be the same as in Construction 1.

$(\text{evk}, \text{evk}') \leftarrow \text{RotKeyGen}$: It runs $\text{TraceKeyGen}(S)$ and $\text{ConvCAKeyGen}(S)$.

$\langle \mathbf{s} \cdot \mathcal{E}_{\text{Rot}_{\text{idx}}(\mathbf{x})}^A \rangle \leftarrow \text{RightRotate}(\text{evk}, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \rangle, \text{idx})$: Given an Type A encoded authenticated share pair, it outputs an authenticated share pair with right-rotated underlying vector by a rotation index idx . Let $\mathbf{m} = (1, 1, \dots, 1, 0, 0, \dots, 0) \in [B]^T$ be the masking vector with idx many ones. Homomorphic rotation is instantiated by the following protocol:

1. **Multiply Masking Vector:** Output

$$\mathcal{E}_{\mathbf{m}}^B \cdot \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \rangle = \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \cdot \mathcal{E}_{\mathbf{m}}^B \rangle.$$

2. **Apply ClearCoeff:** Output

$$\text{ClearCoeff}(\text{evk}, \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \cdot \mathcal{E}_{\mathbf{m}}^B \rangle) = \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x} \odot \mathbf{m}}^C \rangle.$$

3. **Apply ConvCA:** Output

$$\text{ConvCA}(\text{evk}', \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x} \odot \mathbf{m}}^C \rangle) = \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x} \odot \mathbf{m}}^A \rangle.$$

This extracts the first idx of the vector $\mathbf{x}$.

4. **Subtract from** $\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \rangle$: Get the latter half by subtraction.

$$\langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x}}^A \rangle - \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x} \odot \mathbf{m}}^A \rangle = \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x} \odot (1-\mathbf{m})}^A \rangle.$$

5. **Shift and Add:** We compute

$$X^{-\text{idx}} \cdot \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x} \odot (1-\mathbf{m})}^A \rangle + X^{\text{idx}} \cdot \langle \mathbf{s} \cdot \mathcal{E}_{\mathbf{x} \odot \mathbf{m}}^A \rangle,$$

which encrypts the desired result.

The correctness of rotation is automatically checked by the correctness of the individual components. The cost of rotation is roughly the same as the cost of multiplication. Given unpacking, packing, and rotation, addition, and multiplication, we can handle arbitrary arithmetic operations with respect to the encoded vectors.

A.2 Efficiency Analysis

Similar to Section 4.4, we analyze the concrete efficiency of these additional SIMD operations.

Unpacking. Observe that unpacking only requires a monomial by ring element multiplication. In this case, NTT/INTT is not necessary: such multiplication requires a simple coefficient shifting (plus negation) as automorphisms, thus resulting in negligible costs. Thus, unpacking essentially costs $2T$ of ring additions.

Rotation. Rotation is, however, more scheme dependent. Specifically, we need to be able to multiply the two shares, which have different runtimes for different schemes. Thus, one can view rotation as the same cost as a single SIMD multiplication in the corresponding applications.

A.3 Additional Discussions

Mathematical Definition of Trace. Recall the definition of `ClearCoeff`, which sends

$$v = \sum_{i=0}^{N-1} v_i \cdot X^i$$

to

$$\text{ClearCoeff}(v) := \sum_{0 \leq i < N, S|i} v_i \cdot X^i.$$

The detailed mathematical explanation can be written as follows. For a power-of-two integer $m > 1$, the ring $\mathbb{Q}[X]/(X^m+1)$ is identified as an $(\mathbb{Q})$ -extension ring $\mathbb{Q}[\zeta_{2m}]$ for a primitive $2m$ -th root of unity ζ_{2m} . The trace of $\hat{\mathcal{R}} := \mathcal{R} \otimes_{\mathbb{Z}} \mathbb{Q} = \mathbb{Q}[X]/(X^n+1)$ over $\hat{\mathcal{S}} := \mathcal{S} \otimes_{\mathbb{Z}} \mathbb{Q} = \mathbb{Q}[X^S]/(X^n+1)$ is well defined as

$$\text{Tr}_{\hat{\mathcal{R}}/\hat{\mathcal{S}}}(\alpha(X)) = \sum_{\substack{\sigma: \hat{\mathcal{R}} \rightarrow \bar{\mathbb{Q}} \\ \sigma|_{\hat{\mathcal{S}}} = \text{id}}} \sigma(\alpha(X)) = \sum_{i=0}^S \alpha(X^{(2n/S) \cdot i+1}) \in \hat{\mathcal{S}}$$

for any $\alpha(X) \in \hat{\mathcal{R}}$. Such a trace is inherited to ring of integers $\mathcal{R} = \mathcal{O}_{\hat{\mathcal{R}}}$ and $\mathcal{S} = \mathcal{O}_{\hat{\mathcal{S}}}$. Since trace removes all the coefficients except whose degrees are multiples of S , it has been widely used in FHE (e.g. [CDKS21]). Homomorphic trace can be efficiently instantiated with $\log S$ automorphisms.

Understanding Algorithm 3. To briefly understand Algorithm 3, the polynomial at the beginning of step i has nonzero coefficients only for degrees multiple of 2^i :

$$u(X) = \sum_{j=0}^{N/2^i} u_j \cdot X^{2^i \cdot j}.$$

The automorphism φ_i flips the sign of the coefficients whose degrees are odd multiples of 2^i :

$$u'(X) = \sum_{j=0}^{n/2^i} (-1)^j \cdot u_i \cdot X^{2^i \cdot j}.$$

When added with the initial polynomial, this removes the degrees that are odd multiples of 2^i .