How to Compress Garbled Circuit Input Labels, Efficiently

Marian Dietz[1], Hanjun Li[2], and Huijia Lin[2]

[1] ETH Zürich, Switzerland
marian.dietz@inf.ethz.ch
[2] University of Washington, Seattle, WA, USA
{hanjul,rachel}@cs.washington.edu

Abstract. Garbled circuits are a foundational primitive in both theory and practice of cryptography. Given $(\widehat{C}, \mathbf{K}[\mathbf{x}])$, where $\widehat{C}$ is the garbling of a circuit C and $\mathbf{K}[\mathbf{x}] = \{\mathbf{K}[i, x_i]\}_{i\in[|\mathbf{x}|]}$ are the input labels for an input $\mathbf{x}$, anyone can recover $C(\mathbf{x})$, but nothing else about input $\mathbf{x}$.

Most research efforts focus on minimizing the size of the garbled circuit $\widehat{C}$. In contrast, the work by Applebaum, Ishai, Kushilevitz, and Waters (CRYPTO '13) initiated the study of minimizing the cost for transferring the input labels $\mathbf{K}[\mathbf{x}]$. Later improved in a follow-up by Applebaum et al. (STOC '23), the state-of-the-art techniques allow compressing the input labels to the optimal rate of $1 + o(1)$. That is, each input label can be transferred by essentially sending 1 bit. However, existing solutions are computationally expensive, requiring large numbers of public-key operations (such as RSA exponentiation).

In this work, we present an *efficient* input label compression technique based on Ring-LWE. We achieve the same optimal rate of $1 + o(1)$, by making use of additional communication in an offline stage (before the input $\mathbf{x}$ becomes known), a paradigm that has already been explored in prior works. A novel feature of the offline communication in our scheme is that the information sent is either *reusable* or *compressible* using a random oracle, leading to small amortized offline cost $o(|\mathbf{x}|)$. We further demonstrate concrete efficiency through an implementation whose online latency outperforms the naive baseline (which sends all of $\mathbf{K}[\mathbf{x}]$ in the online phase) in a realistic network with a bandwidth of up to 45Mbps. This break-even point could be pushed even further by leveraging the large potential for parallelization of computation.

Finally, we apply our techniques to construct maliciously-secure two-party computation protocols with *succinct online communication*: The online phase starts once the circuit C becomes known, and requires exchanging only $\mathrm{poly}(\lambda)$ bits (independent of $|C|$). After inputs $\mathbf{x}_A, \mathbf{x}_B$ arrive, an additional $|\mathbf{x}_A| + |\mathbf{x}_B| + \mathrm{poly}(\lambda)$ bits need to be sent.

Table of Contents

1 Introduction

Introduced by Yao in the 1980s [Yao82, BHR12], a garbling scheme allows efficiently transforming a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a *garbled circuit* $\widehat{C}$ and a pair of short, λ-bit, keys $(\mathbf{K}[0,i], \mathbf{K}[1,i])$ for every input bit i, where λ is the security parameter. This transformation ensures that $\widehat{C}$, along with the input labels $\mathbf{K}[\mathbf{x}] = (\mathbf{K}[x_1, 1], \cdots, \mathbf{K}[x_n, n])$, reveals only the output $\mathbf{y} = C(\mathbf{x})$ and no other information about the input $\mathbf{x}$. Over the years, Garbled circuits have found a diverse set of applications, in particular for building efficient constant-round multi-party computation protocols [Yao86, BMR90].

Although originally viewed as impractical, tremendous efficiency improvements in the past two decades has brought it to the forefront of practical and deployable cryptography. Much of this research focused on reducing the size of garbled circuits, and hence the communication costs of transferring them, providing a deep understanding on both the theoretical and practical limits [BMR90, NPS99, KS08, PSSW09, KMR14, GLNP15, ZRE15, RR21, GKP+13, HLL23, AJS17, KLW15, BCG+18, JLL23]. However, efficiency related to the other "half" of garbled circuits, namely the input labels, has been relatively neglected so far, and is the focus of this work.

Consider the following simple use case: Many research institutes each want to perform different analyses, described by circuits C_i, over a common sensitive database $\mathbf{x}$, e.g. patient records. Due to the sensitive nature of the data, only the results $C_i(\mathbf{x})$ of approved analysis may be revealed. Garbled circuits offer a non-interactive solution: The data provider can first provide different garbled circuits $\widehat{C}_i$ to the research institutes according to their proposed analysis, and later broadcast a single set of input labels $\mathbf{K}[\mathbf{x}]$, e.g. through a blockchain. Since the dataset $\mathbf{x}$ may be very large, it's desirable to minimize the size of labels $\mathbf{K}[\mathbf{x}]$, which, naively, would require $\lambda \cdot |\mathbf{x}|$ bits. Applebaum, Ishai, Kushilevitz, and Waters [AIKW13] (AIKW) were the first to ask the following natural question: Is rate-λ, i.e., sending λ bits per input bit, optimal?

Optimal Rate $1 + o(1)$. It turns out that rate-λ is far from optimal. AIKW achieves compression of input labels in an *online-offline setting* based on a variety of public key cryptography assumptions including RSA, LWE, or DDH. They showed that after collecting the data $\mathbf{x}$, i.e., in the *online phase*, it suffices to send just $|\mathbf{x}| + \mathrm{poly}(\lambda)$ bits. This requires publishing a potentially large amount of information before seeing $\mathbf{x}$, i.e., in the *offline phase*. Subsequent works [GS18, GOS18] showed how to achieve this under weaker assumptions, Factoring or CDH, with expensive non-black-box use of cryptographic tools. While those results achieve an *optimal* online rate of $1 + o(1)$, their technique comes at the cost of an offline communication $\Omega(|\mathbf{x}| \cdot \lambda)$.

Therefore, a question left open by AIKW is whether the *overall* rate (combining offline and online) can actually be reduced to less than λ. At first sight, it may seem impossible to communicate the labels $\mathbf{K}[\mathbf{x}]$ (which have size $|\mathbf{x}| \cdot \lambda$) using less than $|\mathbf{x}| \cdot \lambda$ bits overall. This intuition is however incorrect, because the $2|\mathbf{x}|$ input keys might be generated in a pseudorandom way that is still sufficient for garbled circuits security.

The above intuition can indeed be realized using a tool called *projective PRG* (pPRG) proposed recently by Applebaum et al. (ABI+) [ABI+23]. A pPRG with public parameter pp is a pseudorandom generator with the new power that the seed sd can be "projected" to a subset T of all output bits, such that, the projected seed sd_T expands to these output bits indexed by T, while keeping the remaining output bits pseudorandom. pPRG with both succinct, i.e., $\mathrm{poly}(\lambda)$-size, public parameters and projected seeds promises to eliminate the offline phase altogether, and simultaneously maintaining $1+o(1)$ online rate. Built upon the techniques in AIKW,

ABI+ [ABI+23] constructed pPRGs with different levels of succinctness based on several public key assumptions.

Summarizing [AIKW13, ABI+23], we now have compression techniques with two levels of communication efficiency. *1)* Optimal overall-rate $1 + o(1)$ is achieved, based either on RSA, or on indistinguishability obfuscation (iO) combined with somewhere statistically binding hash functions (SSBH). This means each label can be transferred by sending essentially a single bit! *2)* Optimal online rate $1 + o(1)$ is attained, albeit with sub-optimal $\Omega(|\mathbf{x}| \cdot \lambda)$ offline cost, based on LWE and DDH. The DDH-based technique can be further improved using bilinear groups, enabling *reusing* the large offline communication, achieving amortized overall-rate $1 + o(1)$ over multiple instances of garbled circuits. See Table 1 for detailed comparison.

	communication	computation
Naive	$\|\mathbf{x}\| \cdot \lambda$	0
DDH ([AIKW13])	$\|\mathbf{x}\| + \|\mathbf{x}\|/\lambda$ + offline: $\|\mathbf{x}\| \cdot \lambda \cdot \log p$	$(\|\mathbf{x}\| \cdot \lambda \cdot \log p)$ Mult_G
LWE ([AIKW13])	$\|\mathbf{x}\| + \|\mathbf{x}\|/\lambda$ + offline: $\|\mathbf{x}\| \cdot \mathrm{poly}(\lambda, n, \log q)$	$(\|\mathbf{x}\| \cdot \lambda \cdot n \cdot \log q)$ Add_q
bilinear DDH ([ABI+23])	$\|\mathbf{x}\| + \|\mathbf{x}\|/\lambda$ ★	$(\|\mathbf{x}\| \cdot \lambda \cdot \log p)$ Mult_G + $\|\mathbf{x}\|$ Bilinear ops
iO+SSBH ([ABI+23])	$\|\mathbf{x}\| + \mathrm{poly}(\log \|\mathbf{x}\|, \lambda)$	$(\|\mathbf{x}\| \cdot \mathrm{poly}(\lambda))$ iO-evals
RSA ([ABI+23])	$\|\mathbf{x}\| + \mathrm{poly}(\lambda)$	$(\|\mathbf{x}\| \cdot \log \|\mathbf{x}\| \cdot \log N)$ Mult_N
Ring-LWE (this work)	$\|\mathbf{x}\| + \mathrm{poly}(\lambda)$ ★	$(\frac{\|\mathbf{x}\| \log \|\mathbf{x}\|}{n})$ $\mathsf{Mult}_{\mathcal{R}_q}$

Table 1: Comparison between techniques for input label compression, in terms of *online communication* and *online computation time*. Constant factors are omitted. $|\mathbf{x}|$ denotes the input length, p is the DDH group order, N is the RSA modulus, n is the Ring-LWE degree / LWE dimension and q is the (Ring-)LWE modulus (which is exponentially large, i.e., $q = 2^{\Theta(\lambda)}$). For computation time, we write Mult to denote the cost of a single multiplication within the ring or modulus indicated in the subscript. When indicated, the scheme requires an additional *offline* step which can be performed without knowing $\mathbf{x}$. We use ★ to denote schemes that require an offline step that only needs to be executed *once*, whose communication therefore vanishes after a sufficiently large number of instances.
We further remark that the costs of the (bilinear) DDH and LWE schemes shown here applies an optimization that splits the input into smaller blocks. See [AIKW13] for details.

Theory vs Practice. Given the importance of practically efficient garbled circuits, it is exciting to try to apply these compression techniques in the wild. Unfortunately, the computational costs of current techniques are extremely high, establishing only feasibility, but not practicality.

Specifically, methods described in [AIKW13, ABI+23] require performing a large number of expensive public key operations, namely, RSA-exponentiation (based on RSA), group exponentiation (DDH), inner products of long vectors over $\mathbb{Z}_p$ (LWE), or pairing (DDH), while methods in [GS18, GOS18] are prohibitively expensive. Take the RSA-based scheme as an example, which is the currently most efficient method. It requires the receiver to run $O(|\mathbf{x}| \cdot \log |\mathbf{x}|)$

RSA-exponentiations[3] (with large exponents) in order to recover the input labels. However, the concrete cost of this is very high: Suppose the input has $100K$ bits i.e., $|\mathbf{x}| \geq 10^5$, and one RSA-exponentiation consumes $1ms$. Then the receiver can only process 60 input labels per second, and this rate decreases even further for larger inputs. All other schemes have even more overhead due to $\lambda \cdot \log p$ or even $\lambda \cdot n \cdot \log q$ factors.

Motivated by the state-of-affairs, the overarching goal of this work is:

Can we transfer input labels in online- or overall-rate less than λ, efficiently*?*

We remark that the state-of-affairs is reminiscent to the current landscape of research on the minimal size of garbled circuits. In theory, optimal $\mathrm{poly}(\lambda, n, m)$-size garbled circuits is feasible (with n, m the input/output length), based on LWE or indistinguishability obfuscation and puncturable PRF [GKP+13, HLL23, AJS17, KLW15, BCG+18, JLL23]. However, these schemes are computationally prohibitive. In practice, optimized versions of Yao's garbled circuits are far more efficient, despite their large communication costs. Similarly, current theoretically optimal input-label compression techniques have not yet brought benefits to practical efficiency. Narrowing the gap is the aim of this work.

Our Results in a Nutshell. We make progress towards the above overarching goal by presenting a new efficient compression technique with optimal overall-rate, $1 + o(1)$, based on Ring-LWE (depending on the choice of ring parameters), in the Random Oracle Model (ROM).

- *Lightweight communication:* The optimal overall-rate $1 + o(1)$ is achieved. While, unlike pPRG-based solutions, our scheme does make use of offline communication, it has the novel feature that information sent in the offline phase is either reusable or lightweight, with amortized offline-rate $o(1)$.
- *Concretely Efficient:* By leveraging Ring-LWE packing, our online computation is dominated by $O(\frac{\log |\mathbf{x}| \cdot \lambda}{n \log q})$ ring operations per input bit (where n is the Ring-LWE degree), a potentially *fractional* number. Indeed, in concrete settings an average of 0.14 ring operations is performed per input bit in our implementation.
- *Implementation:* Our implementation demonstrates the practical efficiency of our method. For transferring the labels of $700K$ input bits, it takes about $0.90\mu s$ per bit for the garbler to compress and $1.77\mu s$ per bit for the evaluator to reconstruct, on a single-core PC.

As a further application, we construct *preprocessing garbled circuits*: relying on an offline phase with $\tilde{O}(\lambda|C|)$ communication, the online phase (that starts once the plaintext circuit C becomes known to both parties) has a reduced communication cost of only $\mathrm{poly}(\lambda)$ bits, independent of the circuit size. These ideas naturally yield maliciously secure two-party computation protocols in the preprocessing model, with optimally *succinct online communication*. Namely, only $\mathrm{poly}(\lambda)$ bits are exchanged after the circuit C becomes available, and $|\mathbf{x}_A| + |\mathbf{x}_B| + \mathrm{poly}(\lambda)$ bits are exchanged after the two parties receive their respective inputs $\mathbf{x}_A$ and $\mathbf{x}_B$. While previous techniques for sublinear-communication 2PC did not rely on preprocessing, all of them require computationally expensive tools, such as, FHE, iO, HSS. Our protocols are much more concretely efficient.

On the Importance of Compressing Input Labels. Before moving on to describing our results in more detail, we want to call for further study on the efficiency of transferring input labels. This topic has so far had a limited history of study, perhaps due to the school of thought that the cost of transferring the input labels is dominated by that of transferring the garbled

[3] This assumes some algorithmic optimization not described in [AIKW13, ABI+23].

circuit. However, there are several strong reasons motivating its study. First, performance optimization entails saving costs in every possible avenue in practice. In big data applictaions the input might even be almost as large as the circuit. Second, in applications that allow preprocessing, the real-time performance becomes mostly dominated by the time required for transferring input labels. Third, the lack of theoretical understanding of this basic and natural question about garbled circuits is unsatisfactory; any progress here may lead to other applications. Finally, we are optimistic that in the future, practical garbled circuits with less than λ bit per gate, or even fixed polynomial size, might become possible. Then, the costs for transferring input labels may even outweigh that of the garbled circuit.

Our Results in More Detail. The key tool in this work, following AIKW techniques, are what we call *Batch-Select* schemes[4]. It enables the sender, Alice, to encode the keys $\mathbf{K}$ in the offline stage:

$$\mathsf{Sel.Enc}_1(\{\mathbf{K}[i,1]-\mathbf{K}[i,0]\}) \to (\mathsf{ct}_1, \mathsf{st}_1)\ , \quad \mathsf{Sel.Enc}_2(\{\mathbf{K}[i,0]\}) \to (\mathsf{ct}_2, \mathsf{st}_2)\ .$$

(Note that the 0-keys $\mathbf{K}[i,0]$ and the key offsets $\mathbf{K}[i,1]-\mathbf{K}[i,0]$ are encrypted separately, which will be convenient later.) Alice publishes the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ and keeps the secret state $\mathsf{st} = (\mathsf{st}_1, \mathsf{st}_2)$. After $\mathbf{x}$ arrives, Alice generates a very succinct key $\mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{x}) \to \mathsf{sk}_\mathbf{x}$ of just $\mathrm{poly}(\lambda)$ size. The pair $(\mathsf{sk}_\mathbf{x}, \mathbf{x})$ enables (partial) decryption of ct, revealing exactly the input labels $\mathsf{Sel.Dec}(\mathsf{ct}, \mathsf{sk}_\mathbf{x}, \mathbf{x}) \to \mathbf{K}[\mathbf{x}] = \{(\mathbf{K}[i,1]-\mathbf{K}[i,0]) \cdot x_i + \mathbf{K}[i,0]\}$, and nothing else. Security is formalized via simulation of ciphertext ct and secret key $\mathsf{sk}_\mathbf{x}$, given only the revealed labels $\mathbf{K}[\mathbf{x}]$.

Batch-Select enables achieving optimal online rate when transferring input labels, because online Alice just sends $\mathbf{x}$ and $\mathsf{sk}_\mathbf{x}$ of length $|\mathbf{x}| + |\mathsf{sk}_\mathbf{x}| = |\mathbf{x}| + \mathrm{poly}(\lambda)$[5]. In this work, we present a concretely efficient Batch-Select scheme with the novel features that part of the offline communication ct_1 can be *reused*, and the rest ct_2 can be *compressed* using the random oracle, leading to small offline-rate $o(1)$. More precisely:

Theorem 1 (Informal, New Batch-Select Scheme). *Let λ be the security parameter. Assuming 2^λ-secure Ring-LWE with dimension n, modulus $\log q = O(\lambda)$, there exists a 2^λ-secure Batch-Select scheme. The scheme has the following asymptotic efficiency when the input length satisfies $|\mathbf{x}| = \Omega(n)$:*

- $|\mathsf{ct}_1| = O(|\mathbf{x}| \log |\mathbf{x}| \cdot \lambda)$. *Ciphertext ct_1 can be reused for $T = \sqrt{q}$ times.* (If the modulus q is super-polynomial, this lets us reuse ct_1 for any polynomial number of times.)
- $|\mathsf{ct}_2| = O(|\mathbf{x}| \cdot \lambda)$ *in the plain model. When ct_2 is used to encrypt* random *keys $\mathbf{K}[i,0]$, its size can be reduced to $O(\frac{\lambda}{q^{1/2-\epsilon} \cdot \log q} \cdot |\mathbf{x}|)$ bits in the ROM for any constant $\epsilon \in (0, \frac{1}{2})$. In particular:*
  - *For sufficiently large polynomial modulus $q = \mathrm{poly}(n, \lambda)$, the rate is $o(1)$.*
  - *For exponentially large modulus $q > 2^{2(1+\epsilon')\lambda}$ for any positive constant $\epsilon' > 0$, ct_2 can be transferred for free (with no communication).*
- $|\mathsf{sk}_\mathbf{x}| = n \log q = \mathrm{poly}(\lambda)$, *i.e., the secret key consists of a single ring element.*
- *The computational efficiency of all algorithms is* quasilinear *in the input length.*

[4] In [AIKW13], this gadget is "special randomized encoding for the *selection function*" and has a slightly different syntax. Nevertheless, it can be used in an analogous way to compress input labels.

[5] In order to achieve input hiding, Alice can apply a one-time pad $\mathbf{r}$ to the input. She would send $\mathbf{x} = \mathbf{x}' + \mathbf{r}$, where $\mathbf{x}'$ is the *actual* input. The one time pad can be removed by modifying the computation to $C'_\mathbf{r}(\mathbf{x}) = C(\mathbf{x} - \mathbf{r})$.

Lightweight Offline Communication The offline communication of our scheme transfers ct_1 and ct_2. Their absolute sizes are large, $O(|\mathbf{x}| \log |\mathbf{x}| \lambda)$ and $O(|\mathbf{x}|\lambda)$, exceeding λ-rate. We crucially rely on the reusability of ct_1 to establish that the amortized communication rate for transferring ct_1 is $o(1)$ or below. While ct_2 is not reusable, it can be compressed in the ROM to rate $o(1)$ or below. Below we give more details on these two optimizations, and summarize the amortized size of ct_1 and compressed size of ct_2 in Table 2.

amortized $\lvert\mathsf{ct}_1\rvert$	amortization	modulus q
$\lambda \cdot \lvert\mathbf{x}\rvert \log \lvert\mathbf{x}\rvert$	$T = 1$	$2^{O(\lambda)}$
$o(1) \cdot \lvert\mathbf{x}\rvert$	$T = \sqrt{q}$	$\mathrm{poly}(\lambda)$
$\mathsf{negl}(\lambda)$	$T = \sqrt{q}$	$\lambda^{\omega(1)}$

(a) Size of ct_1

size $\lvert\mathsf{ct}_2\rvert$	modulus q	$\mathbf{K}[0]$
$\lambda \cdot \lvert\mathbf{x}\rvert$	$2^{O(\lambda)}$	any
$o(1) \cdot \lvert\mathbf{x}\rvert$	$\mathrm{poly}(\lambda)$	random
0	$2^{2(1+\epsilon')\lambda}$	random

(b) Size of ct_2

Table 2: The amount of offline communication (split into $|\mathsf{ct}_1|$ and $|\mathsf{ct}_2|$) required by our batch-select scheme in several different settings, omitting constant factors. On the left-hand side, we show the size of $|\mathsf{ct}_1|$ amortized across T instances (with the same offset $\mathbf{K}[1] - \mathbf{K}[0]$), assuming that q fulfills the stated constraint. On the right-hand side, we show the size of $|\mathsf{ct}_2|$ assuming that q fulfills the stated constraint. In the bottom two rows, the encrypted keys $\mathbf{K}[0]$ cannot be chosen arbitrarily but are generated as random within our batch-select scheme (this requires the ROM). Everywhere we assume the Ring-LWE dimension satisifies $n > \lambda$ and $|\mathbf{x}| = \Omega(n)$.

First, T-time reusability of ct_1 immediately reduces the per-instance cost of sending ct_1 by a factor of T, assuming Alice sends T garbled circuits and corresponding input labels. This amortization only works whenever ct_1 is encrypting the same values across all instances, but this is not an issue when using batch-select as a way to transfer input labels. The reason is that garbled circuits stay secure even when the key offset $\mathbf{K}[i,1] - \mathbf{K}[i,0]$ is repeated for multiple indices i; or even when it is repeated across several garbled circuits. This assumes correlation-robust hash functions [CKKZ12], as in typical garbling with Free-XOR labels [KS08].

Alternatively (when it is not desired to garble several circuits), we can still amortize within the same instance, by treating the input $\mathbf{x}$ as $k \leq T$ shorter inputs $\mathbf{x}_1, \cdots \mathbf{x}_k$ of length $|\mathbf{x}|/k$. Then, the size of ct_1 becomes $O(|\mathbf{x}| \log |\mathbf{x}| \cdot \lambda/k)$. Using an optimized Batch-Select construction (which only yields benefits if not amortization across several instances; and it needs to assume identical secret offsets $\Delta = \mathbf{K}[i,1] - \mathbf{K}[i,0]$ for all i) we can furthermore avoid the $\log |\mathbf{x}|$-factor. Thus, by choosing $k = \omega(\lambda)$, the size of ct_1 has rate $o(1)$, even for just a single garbling instance. While Alice now needs to send k secret keys $\mathsf{sk}_{\mathbf{x}_j}$ in the online phase (for a total size of $k \cdot n \cdot \log q$), this cost is still bounded by $\mathrm{poly}(\lambda)$.

A second optimization applies (in the ROM) whenever the keys $\{\mathbf{K}[i,0]\}$ that are encrypted by ct_2 are *uniformly random* (which is the case for most constructions of garbled circuits). This allows compressing ct_2 by a factor of $\tilde{O}(q^{1/2-\epsilon})$ for any constant $\epsilon \in (0, \frac{1}{2})$. Hence, even with sufficiently large *polynomial* Ring-LWE modulus q, we reduce the size of $|\mathsf{ct}_2|$ to e.g. $|\mathbf{x}|/\lambda$, which implies an $o(1)$-rate. Furthermore, when the modulus is exponentially large $q = 2^{2(1+\epsilon')\lambda}$ for some constant $\epsilon' > 0$, we can "transfer" ct_2 entirely for free, with no communication.

Putting both reusability and ct_2 compression together, the batch-select offline rate approaches 0, while the online rate remains at $1 + o(1)$.

Concretely Efficient Computation The online computation cost of our scheme is dominated by $O(\frac{\log|\mathbf{x}|\cdot\lambda}{n\log q})$ ring operations per input bit. This fractional number is achieved by packing multiple input keys into a single RLWE element.

We demonstrate concrete efficiency based on our batch-select implementation. To prioritize computational cost and simplicity of implementation, we choose a 109-bit modulus, and implement a simplified version of our scheme that requires $6|\mathbf{x}|$ bits of offline communication (and $\mathrm{poly}(\lambda)$ bits of online communication). Clearly, this is not the theoretically optimal overall $o(1)$-rate (see Table 2), but nevertheless a large improvement over the baseline.

Concretely, our implementation has an online computation time of $3\mu s$ per input bit, which we estimate to be 10^4 times faster than the AIKW RSA-based scheme (assuming that each RSA exponentiation takes $1ms$). To demonstrate the practicality of our method, we show (in Table 3) that the computation time (using a single CPU thread with 2.10GHz) from compressing and reconstructing input labels roughly equals the latency of naively transmitting un-compressed input labels over a network that has bandwidth 50 Mbps. But even when the network is faster than this, our method is still valuable e.g. when the input labels need to be broadcast to multiple evaluators, or when size plays more crucial role such as in blockchain-settings. Details on our evaluation can be found in Section 6.

	#NTT	#Mult	Time (μs)
garbler (ours)	0.02	0.01	0.90
evaluator (ours)	0.02	0.12	1.77
Yao-style GC (1 Mbps)	sending 16 bytes		128.0
Yao-style GC (10 Mbps)	sending 16 bytes		12.8
Yao-style GC (50 Mbps)	sending 16 bytes		2.56

Table 3: Concrete online costs (per input bit) of our implementation for input length $|\mathbf{x}| \approx 700$K. The computational costs are dominated by the number of NTT and component-wise vector multiplications (both in a ring of dimension 4096 and < 60-bit modulus). We compare these numbers with the amount of time that would be required for sending *uncompressed* input labels $\mathbf{K}[\mathbf{x}]$ in standard Yao-style garbled circuits.

Application to Two Party Computation (2PC). In the literature there is a wealth of constructions of 2PC protocols. They can be roughly categorized into *i)* ones that are concretely efficient but have large communication linear in the complexity of the computation (e.g., [DPSZ12, LP11, WRK17, DILO22]), *ii)* ones with laconic communication depending only on the input/output lengths, but rely on computationally expensive tools like FHE (e.g., [Gen09, BV11, BGV12, GSW13]), or iO (e.g., [HW15]), etc, and *iii)* ones that enjoy both concrete efficiency and laconic communication, but are restricted to low depth computation like NC_1, using homomorphic secret sharing (e.g., [BGI16]). The latter protocols can be extended to evaluating circuits, but only achieving mildly sublinear communication proportional to $|C|/\log|C|$ or $|C|/\log\log|C|$. Another drawback of protocols in ii) and iii) is that they rely on expensive generic techniques, such as, succinct communication ZK to achieve malicious security. See Appendix A for a more detailed survey of prior sublinear communication 2PC.

Using our compression technique, we explore building concretely efficient malicious 2PC with laconic *online* communication. We start with considering a new notion called preprocessing garbled circuits: By sending information of length proportional to the circuit size $\tilde{O}(\lambda|C|)$ in an

instance-independent stage that can be run before knowing C (depending only on certain meta-information such as circuit size), in the online stage after knowing C, the garbled circuit can be compressed to $\mathrm{poly}(\lambda)$ bits. This is achieved by simply sending a garbled universal circuit $\widehat{U}$ in the instance-independent offline stage, and using our compression technique to transfer input labels for $\widehat{U}$ that allow evaluating the circuit C. Since C is not hidden, the online stage only sends sk_C of $\mathrm{poly}(\lambda)$ bits.

By combining preprocessing garbled circuits and authenticated garbled circuits [WRK17, DILO22], we obtain maliciously secure 2PC with succinct online communication in the ROM based on Ring-LWE: The function-dependent online stage has communication $\mathrm{poly}(\lambda)$, while the input-dependent online stage has communication $\mathrm{poly}(\lambda) + |\mathbf{x}| + |\mathbf{y}|$. The instance-independent preprocessing phase has quasilinear complexity in the circuit size. Importantly, the protocol is concretely efficient, as it only invokes authenticated garbling and our compression technique in a black-box way. (Of course it would also be possible to merge the function- and input-dependent steps into a single phase with communication $\mathrm{poly}(\lambda)+|\mathbf{x}|+|\mathbf{y}|$. However, in practice the function C might become known much earlier than the inputs $\mathbf{x}, \mathbf{y}$, and for concrete efficiency reasons it would make sense to start processing it as soon as possible.)

Prior works [IKM+13, Cou19] have explored using *function-dependent* (instead of instance-independent) preprocessing to reduce the communication after the inputs are known. However, they are in the information-theoretic regime and their main messages have been relatively negative, either the correlated randomness produced by preprocessing is exponentially long, or the online communication is only slightly sublinear, $|C|/O(\log\log|C|)$. See Appendix A.

2 Technical Overview

Our focus is the efficient construction of a *batch-select* scheme Sel over a message space $\mathcal{M}$ that forms a ring. This primitive is a special case of functional encryption: The encryptor, given *two* message vectors $\mathbf{l}_1, \mathbf{l}_2 \in \mathcal{M}^w$, first computes one (large) ciphertext ct. Later, given any selection vector $\mathbf{y} \in \{0,1\}^w$, they compute a (small) decryption key $\mathsf{sk}_\mathbf{y}$. This allows the decryptor (who has the ciphertext ct, key $\mathsf{sk}_\mathbf{y}$, and selection vector $\mathbf{y}$), to compute the evaluation $\mathbf{l}_{\mathsf{res}} := \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$. Here, $\odot$ denotes component-wise vector multiplication. The (simplified) batch-select syntax is as follows:

Encryptor	Decryptor
① $\mathsf{Sel.Enc}(\mathbf{l}_1, \mathbf{l}_2) \to \mathsf{ct}, \mathsf{st}$	③ $\mathsf{Sel.Dec}(\mathsf{sk}_\mathbf{y}, \mathsf{ct}, \mathbf{y}) \to \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$.
② $\mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{y}) \to \mathsf{sk}_\mathbf{y}$	

Security states that no information about $\mathbf{l}_1$ and $\mathbf{l}_2$ beyond $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ is learned. Importantly, we require the size of key $\mathsf{sk}_\mathbf{y}$ to be independent of the message dimension w.

We note that this primitive has already been studied in [AIKW13] (see Table 1), where it was called "randomized encoding for subset functions" (we use the slightly different syntax described above, which is going to be conceptually closer to our construction). We are going to present a new and concretely efficient construction, along with new applications.

2.1 Instantiating Batch-Select

We start by describing a generic way of instantiating the batch-select primitive, given linearly homomorphic encryption (LHE) and linear laconic encryption (LEnc), both working on the batch-select message space $\mathcal{M}$.

Linearly Homomorphic Encryption. The desired batch-select output $\mathbf{l}_{\mathsf{res}}$ for messages $\mathbf{l}_1, \mathbf{l}_2$ and a selection vector $\mathbf{y}$, is visualized on the left-hand side:

$$\underline{\text{Desired}} \qquad\qquad \underline{\text{Achievable with LHE}}$$
$$\begin{pmatrix} \vdots \\ \mathbf{l}_1[i] \\ \vdots \end{pmatrix} \odot \begin{pmatrix} \vdots \\ y_i \\ \vdots \end{pmatrix} + \begin{pmatrix} \vdots \\ \mathbf{l}_2[i] \\ \vdots \end{pmatrix} \quad \text{vs.} \quad \begin{pmatrix} \vdots \\ \mathbf{l}_1[i] \\ \vdots \end{pmatrix} \cdot y + \begin{pmatrix} \vdots \\ \mathbf{l}_2[i] \\ \vdots \end{pmatrix} \tag{1}$$

The difficulty of achieving this comes from *component-wise* multiplication of message $\mathbf{l}_1$ with some long vector $\mathbf{y}$. In particular, if only *vector-scalar* multiplication of $\mathbf{l}_1$ with a single element y were necessary (see right-hand side above), then we could easily achieve this using a key-and-message linearly homomorphic encryption scheme (LHE): The ciphertext returned by $\mathsf{Sel.Enc}$ would contain encryptions $\mathsf{ct}_1^i \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_1, \mathbf{l}_1[i])$ and $\mathsf{ct}_2^i \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_2, \mathbf{l}_2[i])$ for all $i \in [w]$. Given the *succinct* combined key $\mathsf{sk} := \mathsf{sk}_1 \cdot y + \mathsf{sk}_2$, one could decrypt all linearly-combined ciphertext $\mathsf{ct}_{\mathsf{res}}^i := \mathsf{ct}_1^i \cdot y + \mathsf{ct}_2^i$ to obtain $\mathbf{l}_{\mathsf{res}}[i] = \mathbf{l}_1[i] \cdot y + \mathbf{l}_2[i]$.

Linear Laconic Encryption. Our idea for closing the gap between the two terms in Equation 1 is inspired by the recently proposed laconic encryption scheme from [DKL+23] (it is based on Ring-LWE, but for now we describe it for any message space $\mathcal{M}$). We are not going to use their scheme as-is. Instead, we use some of the underlying ideas towards our batch-select construction.

Specifically, our observation is that [DKL+23] implicitly uses a primitive that we denote by LEnc. It allows a sender to encrypt a vector $\mathbf{s} \in \mathcal{M}^w$ of ring elements, and anyone else to locally evaluate the resulting ciphertext ct to obtain the *component-wise multiplication* $\mathbf{s} \odot \mathbf{a}$ between the encrypted vector $\mathbf{s}$ and any chosen vector $\mathbf{a}$. However, this outcome is *masked* by the term $\mathbf{r} \cdot d_{\mathbf{a}}$, for some $\mathbf{r} \in \mathcal{M}^w$ generated during encryption, and a publically computable digest $d_{\mathbf{a}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{a})$ that is also an element in $\mathcal{M}$:

Sender	Receiver
① $\mathsf{LEnc.Enc}(\mathbf{s}) \to (\mathbf{r}, \mathsf{ct})$	② $\mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}) \to \mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a}$

This gadget is helpful, because the evaluation outcome can be seen as "replacing" the difficult-to-achieve component-wise multiplication $\mathbf{s} \odot \mathbf{a}$ by a simple vector-scalar multiplication $\mathbf{r} \cdot d_{\mathbf{a}}$.

Putting It Together. Our batch-select scheme applies the LEnc gadget to message $\mathbf{s} := \mathbf{l}_1$, which will allow the decryptor to obtain $\mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{l}_1 \odot \mathbf{y}$, i.e., the desired component-wise multiplication $\mathbf{l}_1 \odot \mathbf{y}$ masked by the "decryption term" $\mathbf{r} \cdot d_{\mathbf{y}}$. In order to remove this decryption term and simultaneously take care of the second message vector $\mathbf{l}_2$, our scheme additionally produces LHE encryptions of vectors $\mathbf{r}$ and $\mathbf{l}_2$. The encryptor, once selection vector $\mathbf{y}$ is known, computes a *short* combined secret key $\mathsf{sk}_{\mathbf{y}}$ that may be used to evaluate $\mathbf{r} \cdot d_{\mathbf{y}} + \mathbf{l}_2$. Subtracting this from

the LEnc outcome yields $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ as desired. We summarize the scheme as follows.

Encryptor

$$\begin{aligned}
&\textcircled{1}\ \mathsf{Sel.Enc}(\mathbf{l}_1, \mathbf{l}_2) \quad \to \quad \mathsf{ct} := (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1, \mathsf{LHE.ct}_2) \text{ and } \mathsf{st} := (\mathsf{sk}_1, \mathsf{sk}_2)\ , \\
&\qquad\qquad \text{where}\ \ (\mathbf{r}, \mathsf{LEnc.ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{l}_1)\ , \\
&\qquad\qquad\qquad\ \ \mathsf{LHE.ct}_1 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_1, \mathbf{r})\ , \\
&\qquad\qquad\qquad\ \ \mathsf{LHE.ct}_2 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_2, \mathbf{l}_2)\ . \\
&\textcircled{2}\ \mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{y}) \quad \to \quad \mathsf{sk}_{\mathbf{y}} \leftarrow \mathsf{sk}_1 \cdot d_{\mathbf{y}} + \mathsf{sk}_2\ , \\
&\qquad\qquad \text{where}\ \ d_{\mathbf{y}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{y})\ .
\end{aligned}$$

Decryptor

$$\begin{aligned}
&\textcircled{3}\ \mathsf{Sel.Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct}, \mathbf{y}) \quad \to \quad \underbrace{\mathsf{LHE.Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct}_{\mathsf{res}})}_{\mathbf{r} \cdot d_{\mathbf{y}} + \mathbf{l}_2} - \underbrace{\mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \mathbf{y})}_{\mathbf{r} \cdot d_{\mathbf{y}} - \mathbf{l}_1 \odot \mathbf{y}}\ , \\
&\qquad\qquad \text{where}\ \ d_{\mathbf{y}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{y})\ , \\
&\qquad\qquad\qquad\ \ \mathsf{ct}_{\mathsf{res}} := \mathsf{LHE.ct}_1 \cdot d_{\mathbf{y}} + \mathsf{LHE.ct}_2\ .
\end{aligned}$$

As desired, the resulting batch-select decryption key $\mathsf{sk}_{\mathbf{y}}$ is only a single LHE key, independently of w.

Leaving the idealized setting. So far we omitted several details that arise once we replace the generic message space $\mathcal{M}$ by an actual Ring-LWE ring $\mathcal{R}_p$.

First, we ignored Ring-LWE-induced noise. As usual, we can deal with this by multiplying the message vectors $\mathbf{l}_1, \mathbf{l}_2$ with a sufficiently large scaling factor Δ and then work over the ring $\mathcal{R}_q$ for $q = p\Delta$ instead. The final step of the decryptor is to divide and round the output.

We note that this complicates the security proof: since the decryptor will know both the *exact* message $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ as well as its *noisy* variant, some function of the Ring-LWE noise will be leaked. Therefore, we use *noise flooding* to statistically hide the leakage. To keep its cost low, we use a result from [DKL+23], which shows that (for the type of leakage that our scheme produces) it is sufficient to sample the flooding noise from a distribution that is only polynomially larger than the actual Ring-LWE noise. This saves us from an exponentially large Ring-LWE modulus when desired.

Second, naive LHE incurs large noise growth. In particular, the noise resulting resulting from naive multiplication $\mathsf{ct}_1 \cdot d_{\mathbf{y}}$ during LHE evaluation will exceed the modulus q. Therefore, we apply a standard decomposition trick to split ct_1 into $m = \lceil \log_g q \rceil$ ciphertexts encrypting $\mathbf{r}, \mathbf{r} \cdot g, \mathbf{r} \cdot g^2, \ldots$, which can then be multiplied with a decomposition of $d_{\mathbf{y}}$. Performing multiplication in this form will increase existing noise by only a factor of $q^{1/m} \cdot \mathrm{poly}(\lambda)$, which allows us to set $m = O(1)$ without breaking correctness. Applying the same trick inside LEnc, the batch-select ciphertext for message space $\mathcal{M} = \mathcal{R}_p$ has the following size:

$$|\mathsf{LEnc.ct}| = O(w \log w \cdot |\mathcal{R}_q|), \quad |\mathsf{LHE.ct}_1| = O(w \cdot |\mathcal{R}_q|), \quad |\mathsf{LHE.ct}_2| = O(w \cdot |\mathcal{R}_q|).$$

Fortunately, the noise growth of our scheme still allows a large plaintext modulus such as $p \geq q^{1/c}$ for constant c (even with polynomial modulus-to-noise ratio). Therefore, $\log |\mathcal{R}_q| = O(\log |\mathcal{R}_p|)$, implying that a batch-select ciphertext asymptotically has the same size as its plaintext.

Third, for our garbling application, we require batch-select with message space $\mathcal{M} = \mathbb{Z}_p$ instead of ring $\mathcal{R}_p$. This can be achieved easily and efficiently with *packing*: for compatible primes

p, the Chinese Remainder Theorem shows that each $\mathcal{R}_p$ element is isomorphic to $\mathbb{Z}_p^n$ [SV10, GHS12] (where n is the degree of ring elements in $\mathcal{R}$). Thus, whenever $w = \Omega(n)$, we also get a batch-select scheme for message space $\mathcal{M} = \mathbb{Z}_p$ with

$$|\mathsf{LEnc.ct}| = O(w \log w \log p), \quad |\mathsf{LHE.ct}_1| = O(w \log p), \quad |\mathsf{LHE.ct}_2| = O(w \log p).$$

2.2 Making batch-select practical

Overall, batch-select on messages in $\mathcal{M}^w = \mathbb{Z}_p^w$ with $p = \Theta(2^\lambda)$ (as in our application for garbled circuits in Section 2.3) would cost $|\mathsf{ct}| = O(w \log w \lambda)$ bits. For certain settings, we present two optimizations to bring down this cost to essentially 0.

Reusability. Our first observation is that garbled circuit labels under free-XOR type assumptions [KS08] have a special label format $\mathbf{l}_1 = \Delta \cdot \mathbf{1}_w$ (for some fixed $\Delta \in \mathcal{M}$). Even when garbling several circuits, the same label offset may be used for all of them, and hence the message vector $\mathbf{l}_1$ will always be the same. Only $\mathbf{l}_2$ and the selection vector $\mathbf{y}$ change across multiple runs.

The only component of the batch-select ciphertext that depends on $\mathbf{l}_2$ is $\mathsf{LHE.ct}_2$; the other two components $\mathsf{LEnc.ct}$, $\mathsf{LHE.ct}_1$ can be reused. More precisely, we split $\mathsf{Sel.Enc}$ into two parts:

$$\begin{aligned}
\text{Reusable: } ⓵ \mathsf{Sel.Enc}_1(\mathbf{l}_1) \quad &\to \quad \mathsf{ct}_1 := (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1) \text{ and } \mathsf{st}_1 := \mathsf{sk}_1 \ , \\
&\qquad \text{where } \ (\mathbf{r}, \mathsf{LEnc.ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{l}_1) \ , \\
&\qquad\qquad \mathsf{LHE.ct}_1 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_1, \mathbf{r}) \ . \\
\text{Non-reusable: } ⓶ \mathsf{Sel.Enc}_2(\mathbf{l}_2) \quad &\to \quad \mathsf{ct}_2 := \mathsf{LHE.ct}_2 \text{ and } \mathsf{st}_2 := \mathsf{sk}_2 \ , \\
&\qquad \text{where } \ \mathsf{LHE.ct}_2 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_2, \mathbf{l}_2) \ .
\end{aligned}$$

Later, $\mathsf{Sel.KeyGen}$ will be called with both states st_1 and st_2, and $\mathsf{Sel.Dec}$ will be called with both ciphertexts ct_1 and ct_2. The amortized cost for applying batch-select T times becomes $O((\frac{\log w}{T} + 1) w \log p)$. For sufficiently large $T = \Omega(\log w)$, the cost of $|\mathsf{ct}_1|$ vanishes, and only that of $|\mathsf{ct}_2| = O(w \log p)$ remains.

We also note an orthogonal optimization: Whenever $\mathbf{l}_1 = \Delta \cdot \mathbf{1}_w$, we can apply an optimized LEnc construction (Section B.1) that directly yields cost $|\mathsf{LEnc.ct}| = O(w \cdot |\mathcal{R}_q|) = O(w \log p)$ bits, even without amortization. However, using this construction does not allow us to get below the $|\mathsf{ct}_1| = O(w \log p)$ barrier. Only when applying amortization, the following optimization will yield asymptotical advantages.

Compressing Random LHE Ciphertexts. Our second observation is that when transfering common garbled circuit labels, e.g. Yao's garbling, batch-select is used with *completely random* $\mathbf{l}_2$ message vectors. In this case, we may simply sample them within batch-select in the following optimized way: we first sample a *random LHE ciphertext* $\mathsf{LHE.ct}_2$ (this can be done using the random oracle), and reversely derive a message vector $\mathbf{l}_2 \leftarrow \mathsf{LHE.Dec}(\mathsf{sk}_2, \mathsf{LHE.ct}_2)$. It then suffices to publish a seed of λ bits in place of the entire ciphertext $\mathsf{LHE.ct}_2$.

A complication arises as a randomly sampled $\mathsf{LHE.ct}_2$ may not decrypt correctly. Roughly, decryption correctness requires the noise level in $\mathsf{LHE.ct}_2$ to be much smaller than the scaling factor Δ, while a randomly sampled one has uniform noise level in $[0, \Delta)$. This only happens with negligible probability when the modulus $q = 2^{\Theta(\lambda)}$ is sufficiently exponentially large. For general modulus, we utilize rejection sampling to indicate whether a ciphertext is good. It requires sending $|\mathsf{ct}_2| = O(\frac{w \cdot \lambda}{q^{1/2-\varepsilon} \cdot \log q})$ bits given plaintext modulus $p = q^\varepsilon$.

See Section 4.4 for more detail and our security analysis.

Putting It Together. In summary, when applying both discussed optimizations, the cost of batch-select (amortized across T instances with identical $\mathbf{l}_1$ and *uniformly random* $\mathbf{l}_2$ for each instance) becomes

$$|\mathsf{LEnc.ct}| = O(w \log w \log p/T), \quad |\mathsf{LHE.ct}_1| = O(w \log p/T), \quad |\mathsf{LHE.ct}_2^*| = O(w)$$

for general modulus q. With $T = \Omega(\log w \cdot \log p)$, the amortized offline cost will be $O(w)$, which may be much smaller than the plaintext that has size $w \cdot \log p$. With exponential modulus $q = 2^{2(1+\varepsilon')\lambda}$, we even get $|\mathsf{LHE.ct}_2^*| = 0$, and therefore the amortized cost converges towards 0.

2.3 Application: Preprocessing Garbling

Now we sketch a natural application of batch-select: a garbling scheme in which the costly garbling splits into a *function-independent* part, and a subsequent *succinct, function-dependent* part. In other words, the main work can be performed before the circuit is even known.

Our approach is conceptually simple: In a function-independent step ($\mathsf{GarbleU}$), without knowing the function description, the garbler prepares a standard garbling $\widehat{C_U}$ (e.g. using Yao's GC) of a *universal circuit* C_U. This circuit $C_U(f, \mathbf{x})$, when given the description f and input $\mathbf{x}$, returns $f(\mathbf{x})$. We denote the s input keys corresponding to the function description f by $\mathbf{K}^{\mathsf{Fn}} \in \mathcal{K}^{s \times 2}$, and the input keys corresponding to the input $\mathbf{x}$ by $\mathbf{K} \in \mathcal{K}^{|\mathbf{x}| \times 2}$.

In a function-dependent step ($\mathsf{GarbleFunc}$), we only need to make sure that the evaluator obtains the function's input labels

$$\mathbf{K}^{\mathsf{Fn}}[\mathbf{f}] = (\mathbf{K}^{\mathsf{Fn}}[\mathbf{1}] - \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]) \odot \mathbf{f} + \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}] \ ,$$

where $\mathbf{f} \in \{0,1\}^s$ is the bit-representation of function f. We can easily do so, *succinctly*, using batch-select message vectors $\mathbf{l}_1 = \mathbf{K}^{\mathsf{Fn}}[\mathbf{1}] - \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]$ and $\mathbf{l}_2 = \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]$. Besides $\widehat{C_U}$, we place the (large) ciphertext ct into the function-independent garbling (denoted by $\widehat{U} := (\mathsf{ct}, \widehat{C_U})$), and only need to send the short key $\mathsf{sk}_\mathbf{f}$ in the function-dependent step.

We summarize this process below. SG denotes a standard model garbling (such as Yao's garbling scheme). The evaluation function $\mathsf{Eval}((\widehat{C_U}, \mathsf{sk}_\mathbf{f}), f, \mathbf{k}_\mathbf{x})$ takes the function f, the entire garbling $(\widehat{U}, \mathsf{sk}_\mathbf{f})$, and the input labels $\mathbf{k}_\mathbf{x}$ for input $\mathbf{x}$. See Section 5 for details.

$$\mathsf{GarbleU}(\mathbf{K}) \quad \begin{cases} \widehat{C_U} \leftarrow \mathsf{SG.Garble}(C_U, (\mathbf{K}^{\mathsf{Fn}}, \mathbf{K})) \text{ for random keys } \mathbf{K}^{\mathsf{Fn}} \\ \mathsf{ct}, \mathsf{st} \leftarrow \mathsf{Sel.Enc}(\mathbf{K}^{\mathsf{Fn}}[\mathbf{1}] - \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}], \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]) \\ \text{output } \widehat{U} := (\mathsf{ct}, \widehat{C_U}) \text{ and } \mathsf{st} \end{cases}$$

$$\mathsf{GarbleFunc}(\mathsf{st}, \mathbf{f}) \quad \begin{cases} \text{output } \mathsf{sk}_\mathbf{f} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{f}) \end{cases}$$

$$\mathsf{Eval}(f, (\widehat{U}, \mathsf{sk}_\mathbf{f}), \mathbf{k}_\mathbf{x}) \quad \begin{cases} \mathbf{k}_\mathbf{f}^{\mathsf{Fn}} \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_\mathbf{f}, \mathsf{ct}, \mathbf{f}) \\ \text{output } \mathbf{y} \leftarrow \mathsf{SG.Eval}(C_U, \widehat{C_U}, (\mathbf{k}_\mathbf{f}^{\mathsf{Fn}}, \mathbf{k}_\mathbf{x})) \end{cases}$$

The function-dependent garbling size will be $|\mathsf{sk}_\mathbf{f}| = \mathrm{poly}(\lambda)$. Using the optimizations from Section 2.2, the size of function-independent garbling is $|\widehat{U}| = O(|\widehat{C_U}| + |\mathbf{f}| \cdot \lambda)$ for a single instance, and converges towards $|\widehat{C_U}|$ amortized across a large number of instances (in the ROM).

We can naturally transform the garbling scheme above into a 2-party computation protocol with succinct function-dependent cost. One may additionally use batch-select to transfer input labels $\mathbf{k}_\mathbf{x}$ with cost $\mathrm{poly}(\lambda) + |\mathbf{x}|$ in the online phase, instead of the naive cost $|\mathbf{x}| \cdot \lambda$.

2.4 Adaptive Security

In this work we mainly focus on the *selective* simulation security of batch-select, where the attacker must choose all inputs $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(T)})$ before seeing any offline component. Similarly, for preprocessing garbling, all functions $C^{(t)}$ and inputs $\mathbf{x}^{(t)}$ must be chosen statically.

A stronger security requirement would be adaptive security, where the attacker can choose each $\mathbf{x}^{(t)}$ and/or $C^{(t)}$ adaptively, dependent on previously generated components. This notion is important for attaining *maliciously* secure two-party computation protocols, where malicious parties' inputs are not defined at the beginning of the execution. In Appendix F, we show that in the ROM, it is simple to modify our batch-select scheme to become adaptively secure, using similar techniques as in [BHR12]. This requires an adaptive version of error-leakage Ring-LWE (see Definition 4), which follows directly from plain Ring-LWE with exponential modulus-to-noise ratio using noise flooding.

We apply the adaptively secure batch-select to construct a malicious 2-party computation protocol with only $\mathrm{poly}(\lambda)$ bits of communication in the function dependent phase, and $|\mathbf{x}| + |\mathbf{y}| + \mathrm{poly}(\lambda)$ in the online phase.

3 Preliminaries

We denote by λ the security parameter. For two families of distributions $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$, we say that they are 2^λ-indistinguishable, if for any probabilistic adversary $\mathcal{A}$ with running time bounded by $t(\lambda)$, we have

$$|\Pr[\mathcal{A}(X_\lambda) = 1] - \Pr[\mathcal{A}(Y_\lambda) = 1]| \leq \frac{\mathrm{poly}(\lambda)}{2^\lambda} \cdot t(\lambda)$$

for some polynomial $\mathrm{poly}(\lambda)$. We also write this as $X \approx_{\mathrm{c}}^{2^\lambda} Y$. Furthermore, if the *statistical* distance between X and Y is bounded by $\frac{\mathrm{poly}(\lambda)}{2^\lambda}$, then we write $X \approx_{\mathrm{s}}^{2^\lambda}$.

Garbling keys. In our garbling schemes, the i-th bit of an input $\mathbf{x} \in \{0,1\}^{\ell_\mathsf{x}}$ is associated with a pair of *keys* $\mathbf{K}[i,0], \mathbf{K}[i,1] \in \mathcal{K}$. Here, $\mathcal{K}$ is the key space associated with the garbling scheme (e.g., $\mathcal{K} = \mathbb{Z}_2^\lambda$). We call the selected key $\mathbf{K}[i, x_i]$ the *label* of the i-th bit. We write the set of all keys as a matrix $\mathbf{K} \in \mathcal{K}^{\ell_\mathsf{x}\times 2}$. We use short-hands $\mathbf{K}[b] = (\mathbf{K}[1,b], \ldots, \mathbf{K}[\ell_\mathsf{x}, b])$ and $\mathbf{K}[\mathbf{x}] = (\mathbf{K}[1,x_1], \ldots, \mathbf{K}[\ell_\mathsf{x}, x_w])$ (for $b \in \{0,1\}$ and $\mathbf{x} \in \{0,1\}^{\ell_\mathsf{x}}$), to denote the set of all b-keys, and the set of all labels for input $\mathbf{x}$, respectively. Note that both $\mathbf{K}[b]$ and $\mathbf{K}[\mathbf{x}]$ are in $\mathcal{K}^{\ell_\mathsf{x}}$.

3.1 Ring-LWE

For Ring-LWE, we will follow the notation of [DKL+23] for the most part. We recap all necessary definitions and notations.

The m-th *cyclotomic polynomial* is defined as

$$\Phi_m(X) = \prod_{i\in\mathbb{Z}_m^*} (X - \omega_m^i) \in \mathbb{Z}[X] \ ,$$

where ω_m is an m-th root of unity in $\mathbb{C}$. In this work, we consider the polynomial rings $\mathcal{R} = \mathbb{Z}[X]/\Phi_m(X)$, where $\Phi_m(X)$ has a power-of-2 degree $n = \phi(m)$.

Let $\varPhi : \mathcal{R} \to \mathbb{R}^n$ be an embedding of $\mathcal{R}$ in $\mathbb{R}^n$. For example, the coefficient embedding maps $\varPhi(\sum_{i=0}^{n-1} a_i X^i) = (a_0, \ldots, a_{n-1})^T$. We define the *infinity norm* of a ring element $a \in \mathcal{R}$ with respect to its coefficient embedding, i.e., $\|a\|_\infty = \|\varPhi(a)\|_\infty = \max_{i=0,\ldots,d_\mathcal{R}-1} |a_i|$ for $a = \sum_{i=0}^{d_\mathcal{R}-1} a_i X^i$. The *ring expansion factor* of $\mathcal{R}$ is given by $\gamma_\mathcal{R} := \max_{a,b\in\mathcal{R}\setminus\{0\}} \frac{\|a\cdot b\|_\infty}{\|a\|_\infty \cdot \|b\|_\infty}$.

Lemma 1 ([AL21], Proposition 2). *If the cyclotomic $\mathcal{R}$ has degree $n = 2^k$ (i.e., the degree is a power of 2), then the expansion factor is bounded by $\gamma_{\mathcal{R}} \leq n$.*

Given a modulus $q \in \mathbb{N}$, we use $\mathcal{R}_q$ to denote $\mathcal{R}/q\mathcal{R}$. For an element $a \in \mathcal{R}_q$, we use $\|a\|_\infty$ in the ring modulo q, to denote the infinity norm of the element $a \in \mathcal{R}$, where we identify each coefficient $a_i \in \mathbb{Z}_q$ of a with the corresponding representative element in $[\frac{q}{2}, \frac{q}{2}) \subseteq \mathbb{Z}$. We use $\lfloor \frac{a}{\Delta} \rceil$ to denote the ring element $b \in \mathcal{R}_q$, where the i-th coefficient b_i is equal to $\frac{a_i}{\Delta}$, rounded to the nearest integer.

Definition 1 ($\mathsf{LWE}_{\mathcal{R},m,q,\chi}$ Assumption). *The $\mathsf{LWE}_{\mathcal{R},m,q,\chi}$ assumption, for a ring $\mathcal{R}$, number of samples m, modulus q, and error distribution χ over $\mathcal{R}_q$ (all of which are parameterized by λ), states that the following two distributions are indistinguishable:*

$$\left\{ (\mathbf{a}, \mathbf{y}) \middle| \begin{array}{l} \mathbf{a} \leftarrow\$ \mathcal{R}_q^m \\ s \leftarrow\$ \mathcal{R}_q \\ \mathbf{e} \leftarrow\$ \chi^m \\ \mathbf{y} := \mathbf{a} \cdot s + \mathbf{e} \bmod q \end{array} \right\}_\lambda \approx_c^{2^\lambda} \left\{ (\mathbf{a}, \mathbf{y}) \middle| \begin{array}{l} \mathbf{a} \leftarrow\$ \mathcal{R}_q^m \\ \mathbf{y} \leftarrow\$ \mathcal{R}_q^m \end{array} \right\}_\lambda$$

Distributions over $\mathcal{R}$. We will use discrete Gaussians for error distributions in Ring-LWE. Over $\mathbb{R}$, the *continuous* Gaussian distribution with parameter s is defined by the density function $\rho_s(x) = e^{-\pi x^2/s^2}$. Over $\mathbb{Z}$, this yields the *discrete* Gaussian distribution with parameter s by defining the density function $\mathcal{D}_{\mathbb{Z},s}(x) = \frac{\rho_s(x)}{\sum_{x' \in \mathbb{Z}} \rho_s(x')}$. For a ring $\mathcal{R}$ with embedding $\Phi : \mathcal{R} \to \mathbb{R}^n$, the Gaussian distribution $\mathcal{D}_{\mathcal{R},s}$ samples a vector in $\mathbb{R}^n$ using $\mathcal{D}_{\mathbb{Z},s}^n$, and then maps the result back to an element in $\mathcal{R}$. For example, if Φ is the coefficient embedding, $\mathcal{D}_{\mathcal{R},s}$ samples individual coefficients $a_i \leftarrow\$ \mathcal{D}_{\mathbb{Z},s}$, which then yields the ring element $a = \sum_{i=0}^{n-1} a_i X^i$. Gaussians may also be generalized to a *covariance matrix* $\sigma \in \mathbb{R}^{n \times n}$, which allows individual coordinates to be correlated.

To ensure perfect correctness of our constructions, we will actually use *truncated* discrete Gaussians, whose distribution $\overline{\mathcal{D}}_{\mathcal{R},s}$ is based on a security parameter λ. This distribution $\overline{\mathcal{D}}_{\mathcal{R},s}$ is identical to $\mathcal{D}_{\mathcal{R},s}$, except that it rejects the sample if $\|a\|_\infty > \sqrt{\lambda} \cdot s$, and instead continues sampling until it finds an $a \in \mathcal{R}$ with $\|a\|_\infty \leq \sqrt{\lambda} \cdot s$.

Lemma 2. *For a ring $\mathcal{R}$ with degree $n = \mathrm{poly}(\lambda)$, and for any parameter s, the following two distributions are 2^λ-statistically close: $\mathcal{D}_{\mathcal{R},s} \approx_s^{2^\lambda} \overline{\mathcal{D}}_{\mathcal{R},s}$.*

Proof. By [DKL+23, Lemma 2], we have

$$\Pr[\|e\| > \sqrt{\lambda} \cdot s \mid e \leftarrow\$ \mathcal{D}_{\mathcal{R},s}] < 2 \cdot n \cdot e^{-\pi \cdot \lambda} < \frac{\mathrm{poly}(\lambda)}{2^\lambda} .$$

Conditioned on the event given in the probability above not happening, $\overline{\mathcal{D}}_{\mathcal{R},s}$ is identical to $\mathcal{D}_{\mathcal{R},s}$, and hence the statistical distance between these two distributions is bounded by $\frac{\mathrm{poly}(\lambda)}{2^\lambda}$. □

Given any distribution χ, we will use "$\max \chi$" to denote the maximum norm $\|a\|_\infty$ that any value $a \leftarrow \chi$ sampled from this distribution can have (e.g., $\max \overline{\mathcal{D}}_{\mathcal{R},s} = \sqrt{\lambda} \cdot s$).

Error-leakage Ring-LWE. In [DKL+23], the following variant of Ring-LWE (called *error-leakage* Ring-LWE) was introduced. It provides the adversary with some additional information: $\mathcal{A}$ does not only receive the Ring-LWE sample $(\mathbf{a}, \mathbf{y})$, but also some *leakage* $\mathbf{l} = \mathbf{Z} \cdot \mathbf{e} + \overline{\mathbf{e}}$, where $\mathbf{e}$ is the error used to compute $\mathbf{y} = \mathbf{a} \cdot s + \mathbf{e}$, and $\overline{\mathbf{e}}$ is a fresh error from a different distribution $\overline{\chi}$. The *leakage matrix* $\mathbf{Z}$ may be adversarially chosen, but it is restricted to a subset $\mathcal{L} \subset \mathcal{R}^{k \times m}$.

Definition 2 ($\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ **Assumption, [DKL+23]).** *We say that the* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ *assumption (with some set* $\mathcal{L} \subset \mathcal{R}^{k\times m}$*) holds, if for any probabilistic adversary* $\mathcal{A}$ *with running time bounded by* $t(\lambda)$*, we have*

$$\left|\Pr[\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}} = 1] - \Pr[\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}} = 1]\right| \leq \frac{\mathrm{poly}(\lambda)}{2^\lambda} \cdot t(\lambda) \ ,$$

where the $\mathsf{elLWE}^{\mathcal{A},b}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ *experiment is defined as follows.*

1. *The game samples a public parameters* $\mathbf{a} \leftarrow\!\!\$\ \mathcal{R}_q^m$*, and sends* $\mathbf{a}$ *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *outputs a matrix* $\mathbf{Z} \in \mathcal{L}$*.*
3. *The game samples* $\mathbf{e} \leftarrow\!\!\$\ \chi^m$*,* $\overline{\mathbf{e}} \leftarrow\!\!\$\ \overline{\chi}^k$ *and computes the leakage* $\mathbf{l} = \mathbf{Z}\cdot\mathbf{e} + \overline{\mathbf{e}}$*. It sends to* $\mathcal{A}$ *the pair* $(\mathbf{y}, \mathbf{l})$*, where* $\mathbf{y}$ *is computed as follows.*
   - *If* $b = 0$*, sample* $s \leftarrow\!\!\$\ \mathcal{R}_q$ *and define* $\mathbf{y}^T = s\cdot\mathbf{a}^T + \mathbf{e}^T \bmod q$*.*
   - *If* $b = 1$*, sample* $\mathbf{y} \leftarrow\!\!\$\ \mathcal{R}_q^m$ *uniformly at random.*
4. $\mathcal{A}$ *outputs a bit* b'*, which is also the output of the game.*

In [DKL+23], the authors give a hardness result of elLWE under the standard assumption of Ring-LWE. The required parameters for standard Ring-LWE depend on spectral properties guaranteed for all leakage matrices $\mathbf{Z} \in \mathcal{R}^{k\times m}$ in $\mathcal{L}$ (when viewed as a matrix in $\mathbb{R}^{kn\times mn}$). While these propertiese are only explicitly considered in the special case of $k = 1$ in [DKL+23], their proof of elLWE hardness actually works for any matrix $\mathbf{Z} \in \mathcal{R}^{k\times m}$. We will use this more general version, but this also requires us to define the following formalization of a maximal singular value $\sigma_{\max}(\mathcal{L})$. It is based on Lemma 9 of [DKL+23].

Definition 3. *Let* $\mathcal{R}$ *be a ring of degree* n*, and let* $\Phi : \mathcal{R} \to \mathbb{R}^n$ *be an embedding of* $\mathcal{R}$ *in* $\mathbb{R}^n$*. For a vector* $\mathbf{x} \in \mathcal{R}^\ell$*, we write* $\Phi(\mathbf{x}) \in \mathbb{R}^{\ell n}$ *to mean the vector containing all* $\Phi(x_i)$ *for* $\mathbf{x} = (x_1, \ldots, x_\ell)^T$*.*

For a matrix $\mathbf{Z} \in \mathcal{R}^{k\times m}$*, we may define a linear transformation* $\Theta_{\mathbf{Z}} : \mathbb{R}^{mn} \to \mathbb{R}^{kn}$ *with corresponding matrix* $\mathbf{A}_{\mathbf{Z}} \in \mathbb{R}^{kn\times mn}$ *as*

$$\Theta_{\mathbf{Z}}(\mathbf{x}) = \Phi(\mathbf{Z}\cdot\Phi^{-1}(\mathbf{x})) = \mathbf{A}_{\mathbf{Z}}\cdot\mathbf{x} \ .$$

For a set $\mathcal{L} \subset \mathcal{R}^{m\times k}$*, we define the* maximal singular value $\sigma_{max}(\mathcal{L})$ *as*

$$\sigma_{max}(\mathcal{L}) = \max_{\mathbf{Z}\in\mathcal{L}} \sigma_{max}(\mathbf{A}_{\mathbf{Z}}) \ .$$

Theorem 2 ([DKL+23], Theorem 3). *Let* $\epsilon > 0$ *be negligible. Let* $\mathcal{R}$ *be a suitable ring with an embedding as a lattice in* $\mathbb{R}^n$*, and let* $\mathbf{\Sigma}_0$ *be a covariance matrix with* $\sqrt{\mathbf{\Sigma}_0} \geq \eta_\epsilon(\mathcal{R})$*. Let* $\mathcal{L} \subset \mathcal{R}^{k\times m}$ *be an efficiently decidable set. Let* $s, t \geq 2\sqrt{2}$ *be such that*

$$t^2\sigma_{min}(\mathbf{\Sigma}_0) \geq \frac{(s^2+1)(s^2+2)}{s^2}\sigma_{max}(\mathbf{\Sigma}_0)\cdot(\sigma_{max}(\mathcal{L}))^2$$

Now let $\chi = \mathcal{D}_{\mathcal{R},\sqrt{(s^2+1)\mathbf{\Sigma}_0}}$*,* $\overline{\chi} = \mathcal{D}_{\mathcal{R},\sqrt{(t^2+1)\mathbf{\Sigma}_0}}$ *and* $\chi^* = \mathcal{D}_{\mathcal{R},s/2\cdot\sqrt{\mathbf{\Sigma}_0}}$*. Then, assuming that* $\mathsf{LWE}_{\mathcal{R},m,q,\chi^*}$ *is hard,* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ *is also hard. More precisely, if there exists an adversary* $\mathcal{A}$ *with advantage* δ *against* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$*, then there exists an adversary* $\mathcal{A}'$ *with roughly the same running time and adavantage* $\delta - 44\epsilon$ *against* $\mathsf{LWE}_{\mathcal{R},m,q,\chi^*}$*.*

Leakage Considered in This Work. In this work, we will consider the set $\mathcal{L}_{k,m}(\beta) \subset \mathcal{R}^{k\times m}$ of leakage matrices such that all contained ring elements have infinity norm bounded by β (under coefficient embedding). For such matrices, we can show the following bound on the maximal singular values as required by Theorem 2:

Lemma 3 ([DKL+23], Lemma 9 generalized). *Let $\mathcal{R}$ be a cyclotomic ring of degree n that is a power-of-2, and let $\mathbf{Z} \in \mathcal{R}^{k\times m}$ be a matrix, s.t. each coefficient of $\mathbf{Z}$ has $\|\cdot\|_\infty$-norm bounded by β (in the coefficient embedding). Then, the maximal singular value of the matrix $\mathbf{A}_\mathbf{Z} \in \mathbb{R}^{kn\times mn}$ is bounded by $\sigma_{max}(\mathbf{A}_\mathbf{Z}) \leq \beta n\sqrt{km}$. This immediately implies*

$$\sigma_{max}(\mathcal{L}_{k,m}(\beta)) \leq \beta n\sqrt{km} \ .$$

Proof. For any ring element $a \in \mathcal{R}$ with $a = \sum_{i=1}^{n} a_i X^i$ that fulfills $\|\Phi(a)\|_\infty \leq \beta$, and for any $b \in \mathcal{R}$, we have

$$\begin{aligned}\|\Phi(a\cdot b)\|_2 &\leq \sum_{i=1}^{n} |a_i| \cdot \left\|\Phi(X^i \cdot b)\right\|_2 = \sum_{i=1}^{n} |a_i| \cdot \|\Phi(b)\|_2 \leq \beta \cdot \sum_{i=1}^{n} \|\Phi(b)\|_2 \\ &= \beta n \|\Phi(b)\|_2 \ .\end{aligned}$$

Here we used the fact that $X^i \cdot b$ is just a rotation of b which does not change its norm.

We use this result in the following inequality. We split $\mathbf{A}_\mathbf{Z}$ into $k \cdot m$ blocks $\mathbf{A}_\mathbf{Z}[i,j] \in \mathbb{R}^{n\times n}$ for $i \in [k]$, $j \in [m]$ (note that $\mathbf{A}_\mathbf{Z}[i,j]$ is exactly the negacyclic matrix corresponding to the ring element in the i-th row and j-th column of $\mathbf{Z}$). Furthermore, let $\mathbf{x} \in \mathbb{R}^{dm}$ be any vector, which we split into $\mathbf{x}^T = (\mathbf{x}_1^T, \ldots, \mathbf{x}_m^T)^T$, i.e., m smaller vectors $\mathbf{x}_j \in \mathbb{R}^d$, each corresponding to one ring element. Then,

$$\begin{aligned}\|\mathbf{A}_\mathbf{Z} \cdot \mathbf{x}\|_2 &\leq \sqrt{k} \cdot \max_{i\in[k]} \left\| \sum_{j\in[m]} \mathbf{A}_\mathbf{Z}[i,j] \cdot \mathbf{x}_j \right\|_2 \leq \sqrt{k} \cdot \max_{i\in[k]} \sum_{j\in[m]} \|\mathbf{A}_\mathbf{Z}[i,j] \cdot \mathbf{x}_j\|_2 \\ &= \sqrt{k} \cdot \max_{i\in[k]} \sum_{j\in[m]} \left\|\Phi(\mathbf{Z}[i,j] \cdot \Phi^{-1}(\mathbf{x}_j))\right\|_2 \\ &= \beta n\sqrt{k} \cdot \sum_{j\in[m]} \left\|\Phi(\Phi^{-1}(\mathbf{x}_j))\right\|_2 = \beta n\sqrt{k} \cdot \sum_{j\in[m]} \|\mathbf{x}_j\|_2 \\ &\leq \beta n\sqrt{km} \|\mathbf{x}\|_2 \ .\end{aligned}$$

□

We will also consider the following, *sparse*, version of leakage matrices: Let $\mathcal{L}^{\times w}_{k,m}(\beta) \subset \mathcal{R}^{wk\times wm}$ be the set of matrices consisting of w separate blocks (whose infinity norm are each bounded by β) of size $k \times m$ on the diagonal, i.e.,

$$\mathcal{L}^{\times w}_{k,m}(\beta) = \{\mathrm{diag}(\mathbf{Z}_1, \ldots, \mathbf{Z}_w) \mid \mathbf{Z}_i \in \mathcal{L}_{k,m}(\beta)\} \ .$$

Intuitively, a leakage matrix in $\mathcal{L}^{\times w}_{k,m}(\beta)$ says that there are w leakages, each one computed separately from m fresh Ring-LWE samples (but with the same secret key).

Viewing $\mathcal{L}^{\times w}_{k,m}(\beta)$ as a subset of $\mathcal{L}_{wk,wm}(\beta)$ and then directly applying Lemma 3 would incur a factor of w in the upper bound of $\sigma_{\max}(\mathcal{L}_{wk,wm}(\beta))$. Instead, we use the following lemma that shows that this upper bound is independent of w.

Lemma 4. *Let $\mathcal{R}$ be a cyclotomic ring of degree n that is a power-of-2. Then, the maximal singular value of any matrix in $\mathcal{L}_{k,m}^{\times w}(\beta)$ is bounded by*

$$\sigma_{max}(\mathcal{L}_{k,m}^{\times w}(\beta)) \leq \beta n\sqrt{km} \ .$$

Proof. Applying Lemma 3, we get for any $\mathbf{Z} = \mathrm{diag}(\mathbf{Z}_1, \ldots, \mathbf{Z}_w)$ with $\mathbf{Z}_i \in \mathcal{L}_{k,m}(\beta)$ and $\mathbf{x} = (\mathbf{x}_1^T, \ldots, \mathbf{x}_w^T)^T \in \mathbb{R}^{wmd}$ the inequality

$$\|\mathbf{A}_\mathbf{Z} \cdot \mathbf{x}\|_2 = \sqrt{\sum_{i\in[w]} \|\mathbf{A}_{\mathbf{Z}_i} \cdot \mathbf{x}_i\|_2^2} \leq \beta n\sqrt{km} \sqrt{\sum_{i\in[w]} \|\mathbf{x}_i\|_2^2} = \beta n\sqrt{km} \cdot \|\mathbf{x}\|_2 \ .$$

□

Adaptive error-leakage Ring-LWE. In this work, we are additionally interested in an *adaptive* version of error-leakage Ring-LWE. Here, the adversary may choose the leakage matrix $\mathbf{Z} \in \mathcal{L}$ after already seeing the Ring-LWE sample $\mathbf{y}$, and then obtains the leakage. $\mathbf{Z}\cdot\mathbf{e}+\overline{\mathbf{e}}$. Furthermore, this step may be repeated adaptively up to T times, with a separate leakage matrix $\mathbf{Z}^{(t)}$ and noise $\overline{\mathbf{e}}^{(t)}$.

Definition 4 (Adaptive $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}$ Assumption). *We say that the adaptive $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}$ assumption (with some set $\mathcal{L} \subset \mathcal{R}^{k\times m}$) holds, if for any probabilistic adversary $\mathcal{A}$ with running time bounded by $t(\lambda)$, we have*

$$\left|\Pr[\mathsf{a\text{-}elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}} = 1] - \Pr[\mathsf{a\text{-}elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}} = 1]\right| \leq \frac{\mathrm{poly}(\lambda)}{2^\lambda} \cdot t(\lambda) \ ,$$

where the $\mathsf{a\text{-}elLWE}^{\mathcal{A},b}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}$ experiment is defined as follows.

1. *The game samples public parameters $\mathbf{a} \leftarrow\!\!\$\ \mathcal{R}_q^m$ and noises $\mathbf{e} \leftarrow\!\!\$\ \chi^m$. It sends to $\mathcal{A}$ the pair $(\mathbf{a}, \mathbf{y})$, where the Ring-LWE sample $\mathbf{y}$ is computed as follows.*
   - *If $b = 0$, sample $s \leftarrow\!\!\$\ \mathcal{R}_q$ and define $\mathbf{y}^T = s \cdot \mathbf{a}^T + \mathbf{e}^T \bmod q$.*
   - *If $b = 1$, sample $\mathbf{y} \leftarrow\!\!\$\ \mathcal{R}_q^m$ uniformly at random.*
2. *For each $t \in [T]$:*
   - *$\mathcal{A}$ outputs leakage matrix $\mathbf{Z}^{(t)} \in \mathcal{L}$.*
   - *The game samples $\overline{\mathbf{e}}^{(t)} \leftarrow\!\!\$\ \chi^k$ and sends to $\mathcal{A}$ the leakage $\mathbf{l}^{(t)} = \mathbf{Z}^{(t)} \cdot \mathbf{e} + \overline{\mathbf{e}}^{(t)}$.*
3. *$\mathcal{A}$ outputs a bit b', which is also the output of the game.*

Note that the adaptive version is easily implied by standard Ring-LWE whenever the noise $\overline{\mathbf{e}}$ is sampled from a distribution $\overline{\chi}$ that statistically hides $\mathbf{Z} \cdot e$. This variant, called *noise flooding*, has the disadvantage of increasing the bitlength of modulus q by λ.

g-ary gadget vector. In order to constraint noise growth, it will sometimes be necessary to "decompose" an element $a \in \mathcal{R}_q$ (of arbitrary norm) into several ring elements $a_1, \ldots, a_m$ of norm $\|a\|_\infty < g$. We denote the number of elements in the decomposition by $m := \lceil \log_g q \rceil$. By $\mathbf{g}^{-1}$ we denote such a decomposition (written as a column vector), and its inverse is given by the vector

$$\mathbf{g}^T = \left(1\ g\ g^2 \ldots g^{m-1}\right) \in \mathcal{R}_q^m \ .$$

4 Our Main Tool: Batch-Select

In this section, we discuss our new ideas towards an efficient *batch-select* construction. We first give its definition, then define two crucial building blocks LEnc and LHE in Sections 4.1 and 4.2, and present our full construction in Section 4.3. Finally, we discuss our RO-based optimization in Section 4.4.

Definition 5 (Batch-Select). *A* batch-select functional encryption scheme *with abelian message space $\mathcal{M}(\lambda)$ consists of five efficient algorithms:*

- $\mathsf{Sel.Setup}(1^\lambda, 1^w)$ *takes the security parameter λ and dimension w. It outputs public parameters pp, implicitly given as input to all remaining algorithms.*
- $\mathsf{Sel.Enc}_1(\mathbf{l}_1)$ *takes a message vector $\mathbf{l}_1 \in \mathcal{M}^w$, and outputs a ciphertext ct_1 and a state st_1.*
- $\mathsf{Sel.Enc}_2(\mathbf{l}_2)$ *takes a message vector $\mathbf{l}_2 \in \mathcal{M}^w$, and outputs a ciphertext ct_2 and a state st_2.*
- $\mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, \mathbf{y})$ *takes the two states $\mathsf{st}_1, \mathsf{st}_2$ returned by $\mathsf{Enc}_1, \mathsf{Enc}_2$, and a selection vector $\mathbf{y} \in \{0,1\}^w$. It outputs a decryption key $\mathsf{sk}_\mathbf{y}$.*
- $\mathsf{Sel.Dec}(\mathsf{sk}_\mathbf{y}, \mathsf{ct}_1, \mathsf{ct}_2, \mathbf{y})$ *takes the decryption key $\mathsf{sk}_\mathbf{y}$, two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, and the selection vector $\mathbf{y}$. It outputs a message vector $\mathbf{l} \in \mathcal{M}^w$ (which should be $\mathbf{l} = \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$, where $\odot$ denotes component-wise multiplication).*

Correctness. *The scheme* Sel *is correct if for all $\lambda \in \mathbb{N}$, $w \le 2^\lambda$, message vectors $\mathbf{l}_1, \mathbf{l}_2 \in \mathcal{M}^w$, and selection vectors $\mathbf{y} \in \{0,1\}^w$, the following holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_\mathbf{y}, \mathsf{ct}_1, \mathsf{ct}_2, \mathbf{y}) = \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2 \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^w) \\ (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ (\mathsf{ct}_2, \mathsf{st}_2) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2) \\ \mathsf{sk}_\mathbf{y} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, \mathbf{y}) \end{array}\right] = 1.$$

Definition 6 (T-times Simulation Security). *A* Sel *scheme is $T(\lambda)$-times 2^λ-simulation secure if there exists an efficient simulator* $\mathsf{Sel.Sim}$*, such that for all $\{w_\lambda\}_{\lambda\in\mathbb{N}}$, $\{\mathbf{l}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$, $\{\mathbf{l}_{2,\lambda}^{(1)}, \ldots, \mathbf{l}_{2,\lambda}^{(T)}\}_{\lambda\in\mathbb{N}}$, and $\{\mathbf{y}_\lambda^{(1)}, \ldots, \mathbf{y}_\lambda^{(T)}\}_{\lambda\in\mathbb{N}}$, where $w_\lambda \le \mathrm{poly}(\lambda)$, $\mathbf{l}_{1,\lambda}, \mathbf{l}_{2,\lambda}^{(t)} \in \mathcal{M}(\lambda)^{w_\lambda}$, and $\mathbf{y}_\lambda^{(t)} \in \{0,1\}^{w_\lambda}$, the following holds (where we suppress the subscript λ):*

$$\left\{\mathsf{pp}, \mathsf{Sel.Sim}(\mathsf{pp}, \{\mathbf{l}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ \mathbf{l}^{(t)} = \mathbf{l}_1 \odot \mathbf{y}^{(t)} + \mathbf{l}_2^{(t)} \quad \forall t \in [T] \end{array}\right\}_\lambda$$

$$\approx_c^{2^\lambda} \left\{\begin{array}{l} \mathsf{pp}, (\mathsf{ct}_1, \\ \{\mathsf{ct}_2^{(t)}, \mathsf{sk}_\mathbf{y}^{(t)}\}_{[T]}) \end{array} \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2^{(t)}) \quad \forall t \in [T] \\ \mathsf{sk}_\mathbf{y}^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, \mathbf{y}^{(t)}) \quad \forall t \in [T] \end{array}\right\}_\lambda$$

We define two variations of batch-select: a *weak* version in which all entries of the reused message vector $\mathbf{l}_1$ need to be identical (this will allow for a more efficient instantiation), and a *random* version in which the message vector $\mathbf{l}_2$ is required to be uniformly random.

Definition 7 (*Weak* Batch-Select). *A* weak *batch-select scheme is defined identically to Definition 5, except that it is guaranteed that all messages in $\mathbf{l}_1$ are identical. More specifically,* correctness *is only required to hold for messages $\mathbf{l}_1$ of the form $\mathbf{l}_1 = \Delta \cdot \mathbf{1}_w$ for some $\Delta \in \mathcal{M}$.*

*Furthermore, for T-*times 2^λ-simulation security *to hold, it suffices if the indistinguishability in Definition 6 holds for all $\{\mathbf{l}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ where $\mathbf{l}_{1,\lambda} = \Delta \cdot \mathbf{1}_w$ for some $\Delta \in \mathcal{M}$.*

Definition 8 (*Random* Batch-Select). *A* random *batch-select scheme is defined identically to Definition 5, except that the syntax of encryption algorithm* $\mathsf{Sel.Enc}_2$ *changes to:*

- $\mathsf{Sel.Enc}_2()$ *takes no input, and outputs a ciphertext* ct_2*, a message vector* $\mathbf{l}_2 \in \mathcal{M}^w$*, and a state* st_2*.*

The correctness property does not quantify over message vector $\mathbf{l}_2$*. Instead,* $\mathbf{l}_2$ *is generated by* $\mathsf{Sel.Enc}_2()$*. Correctness is only required to hold with overwhelming probability* $1 - \frac{\mathrm{poly}(\lambda)}{2^\lambda}$*.*

We say that a random batch-select scheme satisfies T-times 2^λ-simulation security*, if there exists an efficient simulator* $\mathsf{Sel.Sim}$*, such that for all* $\{w_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{l}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$*, and* $\{\mathbf{y}_\lambda^{(1)}, \ldots, \mathbf{y}_\lambda^{(T)}\}_{\lambda\in\mathbb{N}}$*, where* $w_\lambda \le \mathrm{poly}(\lambda)$*,* $\mathbf{l}_{1,\lambda} \in \mathcal{M}(\lambda)^{w_\lambda}$*, and* $\mathbf{y}_\lambda^{(t)} \in \{0,1\}^{w_\lambda}$*, the following holds (where we suppress the subscript* λ*):*

$$\left\{ \mathsf{pp}, \{\mathbf{l}^{(t)}\}_{[T]}, \mathsf{Sel.Sim}(\mathsf{pp}, \{\mathbf{l}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ \mathbf{l}^{(t)} \leftarrow\!\$\; \mathcal{M}^w \quad \forall t \in [T] \end{array} \right\}_\lambda$$

$$\approx_{\mathrm{c}}^{2^\lambda} \left\{ \begin{array}{l} \mathsf{pp}, \\ \{\mathbf{l}_1 \odot \mathbf{y}^{(t)} + \mathbf{l}_2^{(t)}\}_{[T]}, \\ (\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}, \mathsf{sk}_{\mathbf{y}}^{(t)}\}_{[T]}) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ (\mathsf{ct}_2^{(t)}, \mathbf{l}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Enc}_2() \quad \forall t \in [T] \\ \mathsf{sk}_{\mathbf{y}}^{(t)} \leftarrow \mathsf{Sel}.\mathsf{KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, \mathbf{y}^{(t)}) \quad \forall t \in [T] \end{array} \right\}_\lambda$$

4.1 First building block: LEnc

Our first tool, *linear laconic encryption*, is formally defined in Definition 9. Intuitively, it can be viewed as a functional encryption for (noisily) evaluating $\mathbf{s} \odot \mathbf{a}$ over an encrypted vector $\mathbf{s}$ and a public vector $\mathbf{a}$, but with a specific secret key format and (noisy) decryption procedure as illustrated below.

Encrypt $\mathbf{s}$:	$(\mathsf{st} := \mathbf{r}, \mathsf{ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{s})$,
Generate sk w.r.t. $\mathbf{a}$:	$\mathsf{sk} = \mathbf{r} \cdot d_{\mathbf{a}}, \quad d_{\mathbf{a}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{a})$,
Decrypt $\mathsf{res} = \mathbf{s} \odot \mathbf{a}$:	$\mathsf{res} + \mathrm{noise} = \boldsymbol{\delta} - \mathsf{sk}, \quad \boldsymbol{\delta} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a})$

In particular, the sk is required to be a vector-scalar product between a secret state $\mathbf{r}$ and a public digest of the vector $\mathbf{a}$. During decryption, the ciphertext can first be evaluated to $\boldsymbol{\delta}$ using only the public vector $\mathbf{a}$. The decryption result is then simply $\boldsymbol{\delta} - \mathsf{sk}$.

Definition 9 (Linear Laconic Encryption). *A* LEnc *scheme with associated message space* $\mathcal{R}_q(\lambda)$ *and error bound* $B_{\mathsf{LEnc}}(\lambda)$ *consists of the following efficient algorithms:*

- $\mathsf{LEnc.Setup}(1^\lambda, w)$ *takes the security parameter* λ *and a message dimension* w*. It outputs public parameters* pp*, which is implicitly given as input to all remaining algorithms.*
- $\mathsf{LEnc.Enc}(\mathbf{s})$ *takes output keys* $\mathbf{s} \in \mathcal{R}_q^w$*. It returns input keys* $\mathbf{r} \in \mathcal{R}_q^w$ *and ciphertext* ct*.*
- $\mathsf{LEnc.Digest}(\mathbf{a})$ *takes the database* $\mathbf{a} \in \mathcal{R}_q^w$*. It outputs,* deterministically*, a digest* $d_{\mathbf{a}} \in \mathcal{R}_q$*.*
- $\mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a})$ *takes a ciphertext* ct *and database* $\mathbf{a} \in \mathcal{R}_q^w$*. It outputs* $\boldsymbol{\delta} \in \mathcal{R}_q^w$*.*

Correctness. *The* LEnc *scheme is correct if for all* $\lambda \in \mathbb{N}$*,* $w \le 2^\lambda$ *output keys* $\mathbf{s} \in \mathcal{R}_q^w$*, and database* $\mathbf{a} \in \mathcal{R}_q^w$ *with* $d_{\mathbf{a}} = \mathsf{LEnc.Digest}(\mathbf{a})$*:*

$$\Pr\left[|\boldsymbol{\delta} - (\mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a})| \le B_{\mathsf{LEnc}} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w) \\ \mathbf{r}, \mathsf{ct} \leftarrow \mathsf{LEnc.Enc}(\mathbf{s}) \\ \boldsymbol{\delta} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}) \end{array} \right] = 1 \,.$$

Security, as captured by the following definition, states that the secret vector $\mathbf{s}$ remains hidden even if the noise $\mathbf{e}$ accumulated by evaluation (i.e., the difference between the *actual* outcome $\boldsymbol{\delta}$ and the *expected* outcome $\mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a}$) may be leaked. However, this only holds as long as this leakage is hidden by another noise $\overline{\mathbf{e}}$ sampled from some distribution $\overline{\chi}_{\mathsf{LEnc}}$.

Definition 10 (Simulation security with T-noise leakage). *A* LEnc *scheme is* 2^λ-simulation secure under $T(\lambda)$-noise leakage w.r.t. to noise-hiding distribution $\overline{\chi}_{\mathsf{LEnc}}$ *if there exists an efficient simulator* $\mathsf{LEnc.Sim}$*, s.t. for all* $\{w_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{s}_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{a}_\lambda^{(t)}\}_{\lambda\in\mathbb{N},t\in[T]}$*, where* $w_\lambda \le \mathrm{poly}(\lambda)$*,* $\mathbf{s}_\lambda \in \mathcal{R}_q^{w_\lambda}$*, and* $\mathbf{a}_\lambda^{(t)} \in \mathcal{R}_q^{w_\lambda}$*, the following holds (where we suppress the subscript* λ*):*

$$\left\{ (\mathsf{pp}, \mathsf{ct}, \{\mathbf{e}^{(t)} + \overline{\mathbf{e}}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w) \\ \mathbf{r}, \mathsf{ct} \leftarrow \mathsf{LEnc.Enc}(\mathbf{s}) \\ d_{\mathbf{a}}^{(t)} \leftarrow \mathsf{LEnc.Digest}(\mathbf{a}^{(t)}) \quad \forall t \in [T] \\ \boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)}) \quad \forall t \in [T] \\ \mathbf{e}^{(t)} := \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot d_{\mathbf{a}}^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)}) \quad \forall t \in [T] \\ \overline{\mathbf{e}}^{(t)} \leftarrow\!\$\; \overline{\chi}_{\mathsf{LEnc}}^{w} \quad \forall t \in [T] \end{array} \right\}_\lambda$$

$$\approx_{\mathrm{c}}^{2^\lambda} \left\{ (\mathsf{pp}, \widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)} + \overline{\mathbf{e}}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w) \\ \widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]} \leftarrow \mathsf{LEnc.Sim}(\mathsf{pp}, \{\mathbf{a}^{(t)}\}_{[T]}) \\ \overline{\mathbf{e}}^{(t)} \leftarrow\!\$\; \overline{\chi}_{\mathsf{LEnc}}^{w} \quad \forall t \in [T] \end{array} \right\}_\lambda$$

The following variant of LEnc may be used to construct *weak* batch-select (Definition 7) instead of the full batch-select (Definition 5).

Definition 11 (*Weak* Linear Laconic Encryption). *A* Weak *Linear Laconic Encryption scheme is defined identically to Definition 9, except that it is guaranteed that* all elements of the output key $\mathbf{s}$ are identical. *More specifically,* correctness *is only required to hold for output keys* $\mathbf{s}$ *of the form* $\mathbf{s} = s \cdot \mathbf{1}_w$ *for some* $s \in \mathcal{R}_q$*.*

Furthermore, for 2^λ-simulation security with $T(\lambda)$-noise leakage *to hold, it suffices if the indistinguishability in Definition 10 holds for all* $\{\mathbf{s}_\lambda\}_{\lambda\in\mathbb{N}}$ *where* $\mathbf{s}_\lambda = s_\lambda \cdot \mathbf{1}_w$ *for some* $s_\lambda \in \mathcal{R}_q$*.*

Constructing LEnc. We now present a construction of LEnc over a ring $\mathcal{R}_q$. It is parameterized by

(1) a base g, used for decomposing a ring element into $m = \lceil \log_g q \rceil$ shorter ring elements of norm bounded by g,
(2) the parameter s for the internally used noise distribution $\chi = \mathcal{D}_{\mathcal{R},s}$, and
(3) the parameter $\overline{s}$ for the noise distribution $\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ used to hide the error resulting from LEnc evaluation.

The correctness error will be bounded by

$$B_{\mathsf{LEnc}} \le g \cdot m \cdot \gamma_{\mathcal{R}} \cdot \log_2 w \cdot s \cdot \sqrt{\lambda}\,,$$

where $m = \lceil \log_g q \rceil$, and the construction fulfills simulation security with T-noise leakage w.r.t. noise distribution $\overline{\chi}_{\mathsf{LEnc}}$ under the assumption of $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$.

The construction will essentially be a simplified version of laconic encryption as presented in [DKL+23]. We first briefly recap the important details of the underlying ideas, and then give the full construction.

We will, w.l.o.g., assume that w is a power of 2. If this is not the case, the database may be padded by sufficiently many zeroes. We identify each entry in the database $\mathbf{a}$ with a leaf of the complete binary tree of height $\ell := \log_2 w$. Let us further index the leaves by ℓ-bit bitstrings $\mathsf{ind} \in \{0,1\}^\ell$, and all other nodes on the tree by a prefix $\mathsf{pre} \in \{0,1\}^{<\ell}$. The empty string ϵ denotes the root of the tree.

To compute the digest, we first assign $y_{\mathsf{ind}} := \mathbf{a}[i]$ to the leaves (where ind is the bitstring corresponding to index $i \in [w]$), and then assign a value y_{pre} to each node $\mathsf{pre} \in \{0,1\}^{<\ell}$, computed as a hash $f(\cdot,\cdot)$ of its two children as

$$y_{\mathsf{pre}} := f(y_{\mathsf{pre}\|0}, y_{\mathsf{pre}\|1}) := \mathbf{b}_0^T \cdot (-\mathbf{g}^{-1}(y_{\mathsf{pre}\|0})) + \mathbf{b}_1^T \cdot (-\mathbf{g}^{-1}(y_{\mathsf{pre}\|1})) \ .$$

The root value y_ϵ will represent the LEnc scheme's digest.

Given only a ring element $s \in \mathcal{R}_q$ and an index $\mathsf{ind} \in \{0,1\}^\ell$, $\mathsf{LEnc.Enc}$ needs to output $r_0 \in \mathcal{R}_q$ and a ciphertext ct, such that an evaluator can obtain the result $r_0 \cdot y_\epsilon - s \cdot y_{\mathsf{ind}}$ when given $\mathbf{y}$ and ct. We do so by choosing random $r_0, \ldots, r_{\ell-1} \leftarrow\!\!\$\ \mathcal{R}_q$ and selecting $\mathsf{ct} = (\mathbf{c}_0, \ldots, \mathbf{c}_{\ell-1})$, s.t. for each $i = 0, \ldots, \ell-1$,

$$\mathbf{c}_i \approx \begin{cases} r_i \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + r_{i+1} \cdot \begin{pmatrix} \mathbf{g}^T & \mathbf{0}^T \end{pmatrix} & \text{if } \mathsf{ind}_i = 0 \\ r_i \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + r_{i+1} \cdot \begin{pmatrix} \mathbf{0}^T & \mathbf{g}^T \end{pmatrix} & \text{if } \mathsf{ind}_i = 1 \end{cases} ,$$

where ind_i is the i-th bit of ind. Now, by the way in which the hash function f was constructed, we can evaluate

$$\mathbf{c}_i \cdot \begin{pmatrix} -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|1}) \end{pmatrix} \approx r_i \cdot y_{\mathsf{ind}_{:i}} - r_{i+1} \cdot y_{\mathsf{ind}_{:i+1}} \ ,$$

where $\mathsf{ind}_{:i}$ is the length-i prefix of ind. Therefore, summing up the terms above for all $i = 0, \ldots, \ell-1$ yields $r_0 \cdot y_\epsilon - r_\ell \cdot y_{\mathsf{ind}}$, which is the desired result if the encryptor chooses $r_\ell := s$.

The following construction implements this idea in a *vectorized* fashion, meaning that the ciphertexts above are created for *all* database entries $\mathsf{ind} \in \{0,1\}^\ell$ simultaneously, and evaluation will produce the vector $\mathbf{r}_0 \cdot y_\epsilon - \mathbf{s} \odot \mathbf{y}$.

Construction 1 (Linear Laconic Encryption LEnc).

$\underline{\mathsf{Setup}(1^\lambda, w) \to \mathsf{pp}}$**:** Sample and output Ring-LWE public matrices $\mathsf{pp} := \mathbf{b}_0, \mathbf{b}_1 \leftarrow\!\!\$\ \mathcal{R}_q^m$.

$\underline{\mathsf{Enc}(\mathbf{s}) \to \mathbf{r}, \mathsf{ct}}$**:** Sample Ring-LWE secrets $\mathbf{r}_i \leftarrow\!\!\$\ \mathcal{R}_q^w$ for each layer $i = 0, \ldots, \ell-1$. For notational convenience, set $\mathbf{r}_\ell := \mathbf{s}$. Also sample *truncated* Ring-LWE noises $\mathbf{E}_i \leftarrow\!\!\$\ \overline{\mathcal{D}}_{\mathcal{R},s}^{w \times 2m}$.

Then, for each of the w rows, indexed by $\mathsf{ind} \in \{0,1\}^\ell$, compute for each $i = 0, \ldots, \ell-1$ the ciphertext

$$\mathbf{C}_i[\mathsf{ind}] := \mathbf{r}_i[\mathsf{ind}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \left(\overline{\mathsf{ind}_i} \cdot \mathbf{g}^T \ \ \mathsf{ind}_i \cdot \mathbf{g}^T \right) + \mathbf{E}_i[\mathsf{ind}] \ .$$

We write the results as ℓ matrices $\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1} \in \mathcal{R}_q^{w \times 2m}$.

Return input keys $\mathbf{r} := \mathbf{r}_0$ and ciphertexts $\mathsf{ct} := (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$.

$\underline{\mathsf{Digest}(\mathbf{a}) \to d_{\mathbf{a}}}$**:** Build an $\ell = \log w$-level hash tree as follows, and output the root y_ϵ.

Choose the hash values on the leaves of the tree as $y_{\mathsf{ind}} := \mathbf{a}[\mathsf{ind}]$. (where $\mathbf{a}[\mathsf{ind}]$ denotes the ind-th row of $\mathbf{a}$, where ind is interpreted as the binary integer corresponding to bitstring $\mathsf{ind} \in \{0,1\}^\ell$).

Compute the hash values on the remaining binary tree as

$$y_{\mathsf{pre}} := \begin{pmatrix}\mathbf{b}_0^T & \mathbf{b}_1^T\end{pmatrix} \cdot \begin{pmatrix}-\mathbf{g}^{-1}(y_{\mathsf{pre}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{pre}\|1})\end{pmatrix} \quad \forall \mathsf{pre} \in \{0,1\}^{<\ell} ,$$

and return the digest that is chosen to be the root $d_{\mathbf{a}} := y_\epsilon$.

$\underline{\mathsf{Eval}(\mathsf{ct}, \mathbf{a}) \to \boldsymbol{\delta}}$**:** Parse $\mathsf{ct} := (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$.

As in Digest, compute y_{pre} for all $\mathsf{pre} \in \{0,1\}^{\leq \ell}$ from database $\mathbf{a}$.

Then, for each $\mathsf{ind} \in \{0,1\}^\ell$, compute the result

$$\boldsymbol{\delta}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{C}_i[\mathsf{ind}] \cdot \begin{pmatrix}-\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|1})\end{pmatrix} ,$$

and return the vector $\boldsymbol{\delta}$.

In Appendix B, we formally prove correctness of our LEnc construction above, and show that it fulfills 2^λ-simulation security under T-noise leakage when assuming $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$. In Appendix B.1, we further give a modified version of a *weak* LEnc that is more efficient, but also requires the slightly stronger assumption $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{Tw,2m}(g)}$.

Ciphertext size, running time, and error bounds all depend on the Ring-LWE parameters. For example, we can get the following result (proven in Appendix B.2).

Lemma 5 (LEnc from Ring-LWE). *Assume* $\mathsf{LWE}_{\mathcal{R},O(1),q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$ *holds (for a cyclotomic ring* $\mathcal{R}$ *of degree* $n \geq \lambda$ *that is a power-of-2, modulus* $q \geq n^{15}$*, and modulus-to-noise ratio* $q^{0.9}$*). Then, there exists an* LEnc *scheme over message space* $\mathcal{R}_q$ *with error bound* $B_{\mathsf{LEnc}} \leq \sqrt{q}$ *that is* 2^λ*-simulation secure under* $\sqrt{q}$*-noise leakage w.r.t. to a noise distribution* $\overline{\chi}_{\mathsf{LEnc}}$ *whose values are bounded by* $\max \overline{\chi}_{\mathsf{LEnc}} = 1000\sqrt{q}$*. Furthermore, when encrypting* w*-dimension messages, the components of the scheme have size (in terms of ring elements)*

$$|\mathsf{pp}| = O(1), \quad |\mathsf{ct}| = O(w \log w) ,$$

and its algorithms run in time (in terms of ring multiplication operations)

$$\mathsf{Enc} : O(w \log w), \quad \mathsf{Digest} : O(w), \quad \mathsf{Eval} : O(w \log w).$$

Under the same assumptions, there exists a weak LEnc *scheme secure under* $\frac{\sqrt{q}}{w}$*-noise leakage. Its ciphertext contains only* $|\mathsf{ct}| = O(w)$ *ring elements, and the* Enc *algorithm only requires* $O(w)$ *ring multiplications.*

4.2 Second building block: LHE for Rings

The second tool, *linearly homomorphic encryption*, is a functional encryption for (noisily) evaluating $\mathbf{m}_1 \cdot y + \mathbf{m}_2$ over encrypted vectors $\mathbf{m}_1, \mathbf{m}_2$ and a public element y.

Definition 12 (Noisy Linearly Homomorphic Encryption over Rings). *An* LHE *scheme with associated message space* $\mathcal{R}_q(\lambda)$ *and error bound* $B_{\mathsf{LHE}}(\lambda)$ *consists of the following efficient algorithms:*

- $\mathsf{LHE.Setup}(1^\lambda, 1^w)$ *takes the security parameter* λ *and a message dimension* w*. It outputs public parameters* pp*, which is implicitly given as input to all remaining algorithms.*

– $\mathsf{LHE.Enc}_1(\mathbf{m}_1)$ *takes a message vector* $\mathbf{m}_1 \in \mathcal{R}_q^w$*, and outputs a ciphertext* ct_1 *and a state* st_1*.*
– $\mathsf{LHE.Enc}_2(\mathbf{m}_2)$ *takes a message vector* $\mathbf{m}_2 \in \mathcal{R}_q^w$*, and outputs a ciphertext* ct_2 *and a state* st_2*.*
– $\mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y)$ *takes the two states* $\mathsf{st}_1, \mathsf{st}_2$ *output by* $\mathsf{Enc}_1, \mathsf{Enc}_2$*, and a ring element* $y \in \mathcal{R}_q$*. It outputs a decryption key* sk_y*.*
– $\mathsf{LHE.Dec}(\mathsf{sk}_y, \mathsf{ct}_1, \mathsf{ct}_2, y)$ *takes decryption key* sk_y *and the two ciphertexts* $\mathsf{ct}_1, \mathsf{ct}_2$*. It outputs a decrypted message* $\mathbf{m}_{\mathsf{res}} \in \mathcal{R}_q^w$*.*

Correctness. *The* LHE *scheme is correct if for all* $\lambda, w \in \mathbb{N}$*, messages* $\mathbf{m}_1, \mathbf{m}_2 \in \mathcal{R}_q^w$*, and coefficients* $y \in \mathcal{R}_q$*:*

$$\Pr\left[\left\|\mathbf{m}_{\mathsf{res}} - (\mathbf{m}_1 \cdot y + \mathbf{m}_2)\right\|_\infty \leq B_{\mathsf{LHE}} \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w) \\ \mathsf{ct}_1, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{m}_1) \\ \mathsf{ct}_2, \mathsf{st}_2 \leftarrow \mathsf{LHE.Enc}_2(\mathbf{m}_2) \\ \mathsf{sk}_y \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y) \\ \mathbf{m}_{\mathsf{res}} \leftarrow \mathsf{LHE.Dec}(\mathsf{sk}_y, \mathsf{ct}_1, \mathsf{ct}_2, y) \end{array}\right] = 1\ .$$

Definition 13 (T-times Simulation Security). *An* LHE *scheme is* $T(\lambda)$-time 2^λ-simulation secure *if there exists an efficient simulator* $\mathsf{LHE.Sim}$*, s.t. for all* $\{w_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{m}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{m}_{2,\lambda}^{(1)}, \ldots, \mathbf{m}_{2,\lambda}^{(T)}\}_{\lambda\in\mathbb{N}}$*,* $\{y_\lambda^{(1)}, \ldots, y_\lambda^{(T)}\}_{\lambda\in\mathbb{N}}$*, where* $w_\lambda \leq \mathrm{poly}(\lambda)$*,* $\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda}^{(t)} \in \mathcal{R}_q^w$*, and* $y_\lambda^{(t)} \in \mathcal{R}_q$*, the following holds (where we suppress the subscript* λ*):*

$$\left\{\mathsf{pp}, \mathsf{LHE.Sim}(\mathsf{pp}, \{\mathbf{m}_{\mathsf{res}}^{(t)}, y^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w) \\ \mathbf{m}_{\mathsf{res}}^{(t)} := \mathbf{m}_1 \cdot y^{(t)} + \mathbf{m}_2^{(t)} \quad \forall t \in [T] \end{array}\right\}_\lambda$$

$$\approx_{\mathrm{c}}^{2^\lambda} \left\{\mathsf{pp}, (\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}, \mathsf{sk}_y^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w) \\ \mathsf{ct}_1, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{m}_1) \\ \mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)} \leftarrow \mathsf{LHE.Enc}_2(\mathbf{m}_2^{(t)}) \quad \forall t \in [T] \\ \mathsf{sk}_y^{(t)} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, y^{(t)}) \quad \forall t \in [T] \end{array}\right\}_\lambda$$

Constructing LHE. In this section, we give a construction of LHE over a ring $\mathcal{R}_q$ (see Definition 12). It is parameterized by

(1) a base g, used for decomposing a ring element into $m = \lceil \log_g q \rceil$ shorter ring elements of norm bounded by g,
(2) the parameter s for the noise distribution $\chi = \mathcal{D}_{\mathcal{R},s}$, and
(3) the parameter $\overline{s}$ for the noise distribution $\overline{\chi} = \mathcal{D}_{\mathcal{R},\overline{s}}$.

The correctness error will be bounded by

$$B_{\mathsf{LHE}} \leq (g \cdot m \cdot \gamma_{\mathcal{R}} \cdot s + \overline{s}) \cdot \sqrt{\lambda}\ ,$$

where $m = \lceil \log_g q \rceil$, and the construction fulfills T-times simulation security under the assumption of $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$.

Construction 2 (LHE over rings LHE).

$\underline{\mathsf{Setup}(1^\lambda, 1^w) \to \mathsf{pp}}$**:** Output a public random vector $\mathsf{pp} = \mathbf{a} \leftarrow\!\!\$\ \mathcal{R}_q^w$ of ring elements.

$\underline{\mathsf{Enc}_1(\mathbf{m}_1) \to \mathsf{ct}_1, \mathsf{st}_1}$**:** Sample Ring-LWE secrets $\mathbf{s}_1 \leftarrow\$ \mathcal{R}_q^m$ and *truncated* noises $\mathbf{E} \leftarrow\$ \overline{\mathcal{D}}_{\mathcal{R},s}^{w\times m}$. Output a ciphertext

$$\mathsf{ct}_1 := \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \in \mathcal{R}_q^{w\times m} ,$$

together with state $\mathsf{st}_1 = \mathbf{s}_1$.

$\underline{\mathsf{Enc}_2(\mathbf{m}_2) \to \mathsf{ct}_2, \mathsf{st}_2}$**:** Sample a Ring-LWE secret $s_2 \leftarrow\$ \mathcal{R}_q$ and *truncated* noises $\overline{\mathbf{e}} \leftarrow\$ \overline{\mathcal{D}}_{\mathcal{R},\bar{s}}^{w}$. Output a ciphertext

$$\mathsf{ct}_2 := \mathbf{a} \cdot s_2 + \mathbf{m}_2 + \overline{\mathbf{e}} \quad \in \mathcal{R}_q^{w} ,$$

together with state $\mathsf{st}_2 = \mathbf{s}_2$.

$\underline{\mathsf{KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y) \to \mathsf{sk}_y}$**:** Parse the states $\mathsf{st}_1 = \mathbf{s}_1 \in \mathcal{R}_q^m$ and $\mathsf{st}_2 = s_2 \in \mathcal{R}_q$. Output a decryption key

$$\mathsf{sk}_y := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2 \quad \in \mathcal{R}_q .$$

$\underline{\mathsf{Dec}(\mathsf{sk}_y, \mathsf{ct}_1, \mathsf{ct}_2, y) \to \mathbf{m}_{\mathsf{res}}}$**:** First combine the ciphertexts into

$$\begin{aligned} \mathsf{ct}_{\mathsf{res}} &:= \mathsf{ct}_1 \cdot \mathbf{g}^{-1}(y) + \mathsf{ct}_2 \quad \in \mathcal{R}_q^{w} , \\ // \text{ s.t. } \mathsf{ct}_{\mathsf{res}} &= \mathbf{a} \cdot (\mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2) + (\mathbf{m}_1 \cdot \mathbf{g}^T \cdot \mathbf{g}^{-1}(y) + \mathbf{m}_2) + \text{noise} \end{aligned}$$

Then, use sk_y to decrypt and return $\mathbf{m}_{\mathsf{res}} = \mathsf{ct}_{\mathsf{res}} - \mathbf{a} \cdot \mathsf{sk}_y$, which is supposed to equal $\mathbf{m}_{\mathsf{res}} = \mathbf{m}_1 \cdot y + \mathbf{m}_2 +$ noise.

In Appendix C, we formally prove correctness of our LHE construction above, and show that it fulfills T-time 2^λ-simulation security when assuming $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$.

Ciphertext size, running time, and error bounds all depend on the Ring-LWE parameters. For example, we can get the following result (proven in Appendix C.1).

Lemma 6 (LHE from Ring-LWE). *Assume* $\mathsf{LWE}_{\mathcal{R},w,q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$ *holds (for a cyclotomic ring* $\mathcal{R}$ *of degree* $n \geq \lambda$ *that is a power-of-2, modulus* $q \geq n^{15}$*, and modulus-to-noise ratio* $q^{0.9}$*). Then, there exists an* LHE *scheme over message space* $\mathcal{R}_q$ *with error bound* $B_{\mathsf{LHE}} \leq 10\sqrt{q}$ *that is* $\sqrt{q}$*-times* 2^λ*-simulation secure. Furthermore, when encrypting* w*-dimension messages, the components of the scheme have size*

$$|\mathsf{pp}| = O(w), \quad |\mathsf{ct}_1| = O(w), \quad |\mathsf{ct}_2| = w, \quad |\mathsf{sk}| = 1 ,$$

and its algorithms run in time (in terms of ring multiplications operations)

$$\mathsf{Enc}_1 : O(w), \quad \mathsf{Enc}_2 : O(w), \quad \mathsf{KeyGen} : O(1), \quad \mathsf{Dec} : O(w).$$

4.3 Instantiating Batch-Select from LEnc and LHE

We now present a construction of *batch-select* (Definition 5) based on LEnc and LHE. We will support the message space $\mathcal{M} = \mathbb{Z}_p$, by working over a cyclotomic ring $\mathcal{R}$ of a power-of-2 degree n and composite modulus $q = p \cdot \Delta$. To ensure correctness, Δ must fulfill $\Delta/2 > B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}}$ (where B_{LEnc} and B_{LHE} denote the maximum error incurred by LEnc and LHE, and $\overline{\chi}_{\mathsf{LEnc}}$ is the distribution of noise required to hide any noise leakage from LEnc; see Definition 10).

Plaintext encoding. We use the Chinese Remainder Theorem (CRT) to pack n elements of $\mathbb{Z}_p$ elements into a single plaintext in $\mathcal{R}_p$. Specifically, in our choice of the ring $\mathcal{R} = \mathbb{Z}[X]/\Phi(m)$,

the m-th cyclotomic polynomial (of degree $n = \phi(m)$) splits mod p if $p \equiv 1 \bmod m$, i.e., $\Phi(m) = F_1 \cdot \ldots \cdot F_n$ over $\mathbb{F}_p$, where F_i are degree 1 polynomials. By CRT, the plaintext space is isomorphic to n times $\mathbb{Z}_p$:

$$\mathcal{R}_p = \mathbb{Z}_p[X]/\phi(m) \cong \mathbb{Z}_p[X]/F_1(X) \times \ldots \times \mathbb{Z}_p[X]/F_n(X) \cong \mathbb{Z}_p^n.$$

We write $\mathrm{crt} : \mathbb{Z}_p^n \to \mathcal{R}_p$ to denote a ring isomorphism that "encodes" n plaintexts from the space $\mathbb{Z}_p$ as a single ring element in $\mathcal{R}_p$. Applying CRT to the plaintext space of Ring-LWE is a common idea in the literature [SV10, GHS12]. We refer readers to [GHS12] for more details of the underlying algebra.

Using packing, we obtain an efficient encoding Encode that maps w-dimensional messages $\mathbf{l} \in \mathcal{M}^w$ to $w' = \lceil \frac{w}{n} \rceil$-dimensional ring elements in $\mathcal{R}_p^{w'}$. Importantly, this mapping fulfills the identity $\mathsf{Encode}(\mathbf{a} \odot \mathbf{b}) = \mathsf{Encode}(\mathbf{a}) \odot \mathsf{Encode}(\mathbf{b})$ for any vectors $\mathbf{a}, \mathbf{b} \in \mathcal{M}^w$, and similarly for addition. Note that while Encode maps to $\mathcal{R}_p^{w'}$, in our construction we will treat the resulting vectors as elements in the larger space $\mathcal{R}_q^{w'}$.

Construction 3 (Batch-Select Sel).

$\underline{\mathsf{Setup}(1^\lambda, 1^w) \to \mathsf{pp}}$**:** Run $\mathsf{pp}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{w'})$, $\mathsf{pp}_{\mathsf{LEnc}} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w')$ with $w' = \lceil \frac{w}{n} \rceil$ as defined above, and output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{LHE}}, \mathsf{pp}_{\mathsf{LEnc}})$.

$\underline{\mathsf{Enc}_1(\mathbf{l}_1) \to \mathsf{ct}_1, \mathsf{st}_1}$**:** Encode messages $\mathbf{l}_1 \in \mathcal{M}^w = \mathbb{Z}_p^w$ as a vector $\widehat{\mathbf{l}}_1 = \mathsf{Encode}(\mathbf{l}_1) \in \mathcal{R}_p^{w'}$, and then compute the laconic encryption ciphertext $\mathsf{LEnc.ct}$ for the resulting vector of ring elements:

$$\mathbf{r}, \mathsf{LEnc.ct} \leftarrow \mathsf{LEnc.Enc}(\widehat{\mathbf{l}}_1 \cdot \Delta)$$

Encrypt the corrosponding LEnc keys $\mathbf{r}$ using the *resuable* LHE component:

$$\mathsf{LHE.ct}_1, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{r})$$

Return ciphertext $\mathsf{ct}_1 := (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$ and state st_1.

$\underline{\mathsf{Enc}_2(\mathbf{l}_2) \to \mathsf{ct}_2, \mathsf{st}_2}$**:** Encode messages $\mathbf{l}_2 \in \mathcal{M}^w = \mathbb{Z}_p^w$ as a vector $\widehat{\mathbf{l}}_2 = \mathsf{Encode}(\mathbf{l}_2) \in \mathcal{R}_p^{w'}$. In addition, sample noise $\overline{\mathbf{e}}_{\mathsf{LEnc}} \leftarrow\!\!\$\ \overline{\chi}_{\mathsf{LEnc}}^{w'}$ (recall that LEnc-security only holds as long as its leakage is hidden by noise sampled from $\overline{\chi}_{\mathsf{LEnc}}$).

Then, encrypt $\widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}}$ using the *non-reusable* LHE component.

$$\mathsf{LHE.ct}_2, \mathsf{st}_2 \leftarrow \mathsf{LHE.Enc}_2(\widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}})$$

Return ciphertext $\mathsf{ct}_2 := \mathsf{LHE.ct}_2$ and state st_2.

$\underline{\mathsf{KeyGen}(\mathsf{st}_1, \mathsf{st}_2, \mathbf{y}) \to \mathsf{sk}_{\mathbf{y}}}$**:** Encode selection vector $\mathbf{y} \in \{0,1\}^w \subseteq \mathbb{Z}_p^w$ as a vector $\widehat{\mathbf{y}} = \mathsf{Encode}(\mathbf{y}) \in \mathcal{R}_p^{w'}$. Then, compute the digest $d_{\widehat{\mathbf{y}}} \leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}})$, and return the LHE secret key $\mathsf{sk}_{\mathbf{y}} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, d_{\widehat{\mathbf{y}}})$.

$\underline{\mathsf{Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct}_1, \mathsf{ct}_2, \mathbf{y}) \to \mathbf{l}_{\mathbf{y}}}$**:** As in KeyGen, encode selection vector $\mathbf{y} \in \{0,1\}^w$ as $\widehat{\mathbf{y}} = \mathsf{Encode}(\mathbf{y}) \in \mathcal{R}_p^{w'}$ and compute its digest $d_{\widehat{\mathbf{y}}} \leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}})$. Parse $\mathsf{ct}_1 = (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$, $\mathsf{ct}_2 = \mathsf{LHE.ct}_2$.

To decrypt, first evaluate the LHE to obtain res':

$$\mathsf{res}' \leftarrow \mathsf{LHE.Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{LHE.ct}_1, \mathsf{LHE.ct}_2, d_{\widehat{\mathbf{y}}})$$

$$// \text{ s.t. } \mathsf{res}' = \mathbf{r} \cdot d_{\widehat{\mathbf{y}}} + \widehat{\mathbf{l}}_2 \cdot \Delta + \text{noise}$$

Now we will evaluate the laconic encryption to obtain $\mathbf{r} \cdot d_{\widehat{\mathbf{y}}} - (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}}) \cdot \Delta$ (plus noise). Subtracting this from res' yields the encoding res of the desired message $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$:

$$\mathsf{res} := \mathsf{res}' - \mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}})$$
$$// \text{ s.t. } \mathsf{res} = (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}}) \cdot \Delta + \widehat{\mathbf{l}}_2 \cdot \Delta + \text{noise}$$

Finally, return the decoded result $\mathsf{Encode}^{-1}(\lfloor \frac{\mathsf{res}}{\Delta} \rceil)$.

We defer proving correctness and T-times simulation security (which holds when both LHE and LEnc are T-times simulation secure) to Appendix D.

Our construction above works over message space $\mathbb{Z}_p$, where p depends on Ring-LWE parameters. However, in our garbling application, we require a message space of size at least 2^λ (so that messages may hold input labels). To handle this even with small Ring-LWE modulus, we may easily turn our construction into a batch-select scheme with message space $\mathbb{Z}_p^\ell$ (s.t. $p^\ell \geq 2^\lambda$) by splitting each message across ℓ slots. Then, by combining this with underlying LEnc and LHE (Lemmas 5 and 6), we can e.g. construct a batch-select scheme with the following parameters (proven in Appendix D.2):

Theorem 3 (Batch-Select). *Assume $\mathsf{LWE}_{\mathcal{R},\infty,q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$ holds (for a cyclotomic ring $\mathcal{R}$ of degree $n \geq \lambda$ that is a power-of-2, modulus $q \geq n^{15}$ with $q \geq 2^{60}$ of length $\log q = O(\lambda)$, and modulus-to-noise ratio $q^{0.9}$). Furthermore, suppose modulus q is a composite number $q = p \cdot \Delta$ for some plaintext modulus $p \leq q^{1/4}$ (with $p = q^{\Theta(1)}$) that is an "NTT friendly" prime, i.e., n divides $(p-1)$. Then, there exists a batch-select scheme with message space $\mathcal{M} := \mathbb{Z}_p^\ell$ of size $|\mathcal{M}| \geq 2^\lambda$ that is $\sqrt{q}$-times 2^λ-simulation secure. When encrypting w-dimensional messages (with $w \geq \Omega(n)$), the scheme's components have size*

$$|\mathsf{pp}| = O(w \cdot \lambda), \quad |\mathsf{ct}_1| = O(w \log w \cdot \lambda), \quad |\mathsf{ct}_2| = O(w \cdot \lambda), \quad |\mathsf{sk}_{\mathbf{y}}| = \mathrm{poly}(\lambda)\ ,$$

and its algorithms run in time (in terms of ring multiplication operations):

$$\mathsf{Enc}_1 : O\left(\frac{w \log w}{n}\right), \quad \mathsf{Enc}_2 : O\left(\frac{w}{n}\right), \quad \mathsf{KeyGen} : O\left(\frac{w}{n}\right), \quad \mathsf{Dec} : O\left(\frac{w \log w}{n}\right)\ .$$

A weak *batch-select scheme with $\frac{\sqrt{q}}{w}$-times 2^λ-simulation security can be constructed under the same parameters. Then, ciphertext ct_1 has only size $|\mathsf{ct}_1| = O(w \cdot \lambda)$, and the Enc_1 algorithm only requires $O(\frac{w}{n})$ ring multiplications.*

The latter also implies a weak batch-select scheme with 1-time 2^λ-simulation security whenever $q \geq \omega((w\lambda)^2)$, where the size of ct_1 reduces to $|\mathsf{ct}_1| = o(w)$.

4.4 Sending encryptions of random strings with a random oracle

We now present an optimization that only allows constructing a *random* batch-select scheme in the ROM. For this construction, we cannot use LHE as a black-box anymore. Instead, we need to look inside $\mathsf{LHE.Enc}_2$, to revisit the way this algorithm is encrypting the messages $\widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}}$. Denote by $\mathbf{a}$ the public parameters used inside LHE, and by s_2 and $\overline{\mathbf{e}} \leftarrow \overline{\mathcal{D}}^w_{\mathcal{R},\bar{s}}$ the secrets and noises sampled inside $\mathsf{LHE.Enc}_2$. Then, the ciphertext $\mathsf{LHE.ct}_2$ will have the form

$$\mathsf{LHE.ct}_2 = \underbrace{\mathbf{a} \cdot s_2 + \overline{\mathbf{e}}_{\mathsf{LEnc}} + \overline{\mathbf{e}}}_{\mathbf{c}} + \widehat{\mathbf{l}}_2 \cdot \Delta$$

However, regardless of the exact way this ciphertext is generated, we note that for the batch-select output to be correct it already suffices if $\mathsf{LHE.ct}_2 = \mathbf{a} \cdot s_2 + \widehat{\mathbf{l}}_2 \cdot \Delta + \mathbf{e}$ for *some* error $\mathbf{e}$ with

$$\|\mathbf{e}\|_\infty + B_{\mathsf{LEnc}} + (B_{\mathsf{LHE}} - \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}) < \Delta/2 \,. \tag{2}$$

This is because the result res computed by the batch-select decryptor Dec will be $\widehat{\mathbf{l}}_2 \cdot \Delta + \mathbf{e}$ + LEnc-noise + LHE-noise (note that we do not need to accomodate for the $\leq \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ noise resulting from $\overline{\mathbf{e}}$ as it is already included in $\mathsf{LHE.ct}_2$; hence LHE evaluation contributes at most another $B_{\mathsf{LHE}} - \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ noise).

Modifying batch-select. Our idea to reduce the size of $\mathsf{LHE.ct}_2$ is to reverse the order of sampling. Instead of calculating $\mathsf{LHE.ct}_2$ based on a random message $\widehat{\mathbf{l}}_2$, we sample some $\mathsf{LHE.ct}_2^*$ using the random oracle H (with image $\mathbb{Z}_q$), and reversely determine $\widehat{\mathbf{l}}_2$. We replace $\mathsf{Sel.Enc}_2$ by the following process (note that it does not take any input except pp, since we are constructing a *random* batch-select scheme):

(1) As before, sample noise $\overline{\mathbf{e}}_{\mathsf{LEnc}} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'}$ (used to hide the noise leakage resulting from LEnc evaluation). Further sample LHE secret $s_2 \leftarrow\!\!\$\ \mathcal{R}_q$ and noise $\overline{\mathbf{e}} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}^{w'}$ (this was previously done inside $\mathsf{LHE.Enc}_2$). Use these values to compute $\mathbf{c} := \mathbf{a} \cdot s_2 + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}$ (where $\mathbf{a}$ are the LHE public parameters). In addition, sample an RO-seed $\mathsf{seed} \leftarrow\!\!\$\ \{0,1\}^\lambda$.

(2) We now sample the ciphertext $\mathsf{LHE.ct}_2^*$ in the following way (where we abuse notation to denote by $\mathbf{z}[i,j] \in \mathbb{Z}_q$ the j-th coefficient of the i-th entry in a vector $\mathbf{z} \in \mathcal{R}_q^{w'}$ of ring elements): for all indices $i \in [w']$ and $j \in [n]$, find the smallest integer $d_{i,j} \in \mathbb{N}$ s.t.

$$\|H(\mathsf{seed}, i, j, d_{i,j}) - \mathbf{c}[i,j] \bmod \Delta\|_\infty + B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} < \Delta/2 \,. \tag{3}$$

(We can view this step as a form of rejection sampling: we continue increasing $d_{i,j} = 0$ until it reaches a value for which the sampled value has sufficiently small error.)
Choose $\mathsf{LHE.ct}_2^*[i,j] := H(\mathsf{seed}, i, j, d_{i,j})$.

(3) Finally, we reversely compute messages $\widehat{\mathbf{l}}_2$ from $\mathsf{LHE.ct}_2^*$ by computing

$$\widehat{\mathbf{l}}_2 = \left\lfloor \frac{\mathsf{LHE.ct}_2^* - \mathbf{c}}{\Delta} \right\rceil \in \mathcal{R}_p^{w'} . \tag{4}$$

(4) Output ciphertext $\mathsf{ct}_2 := (\mathsf{seed}, (d_{i,j})_{i \in [w'], j \in [n]})$, labels $\widehat{\mathbf{l}}_2$, and state $\mathsf{st}_2 := s_2$.

We modify $\mathsf{Sel.Dec}$ in the following way:

(1) Before doing anything else, recover the ciphertext $\mathsf{LHE.ct}_2^*$ given only $\mathsf{ct}_2 := (\mathsf{seed}, (d_{i,j})_{i \in [w'], j \in [n]})$, by computing $\mathsf{LHE.ct}_2^*[i,j] := H(\mathsf{seed}, i, j, d_{i,j})$.

(2) Then, continue as before.

Correctness. We verify that correctness still holds: fix any ciphertext $\mathsf{LHE.ct}_2^*$ generated by the process above, and denote by $\mathbf{e} \in \mathcal{R}_q^{w'}$ the unique vector for which $\mathsf{LHE.ct}_2^* = \mathbf{a} \cdot s_2 + \widehat{\mathbf{l}}_2 \cdot \Delta + \mathbf{e}$. Note that $\mathsf{LHE.ct}_2^* - \mathbf{c} - \widehat{\mathbf{l}}_2 \cdot \Delta = (\mathsf{LHE.ct}_2^* - \mathbf{c}) \bmod \Delta$, see Equation 4. Furthermore, by Equation 3 we have $\|(\mathsf{LHE.ct}_2^* - \mathbf{c}) \bmod \Delta\|_\infty < \Delta/2 - B_{\mathsf{LEnc}} - B_{\mathsf{LHE}} - \max \overline{\chi}_{\mathsf{LEnc}}$. By definition of $\mathbf{c} = \mathbf{a} \cdot s_2 + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}$, we thus get

$$\begin{aligned}\|\mathbf{e}\|_\infty = \|\mathsf{LHE.ct}_2^* - \mathbf{c} - \widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}\|_\infty &\leq \|(\mathsf{LHE.ct}_2^* - \mathbf{c}) \bmod \Delta\|_\infty + \underbrace{\|\overline{\mathbf{e}}\|_\infty}_{\leq \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}} + \underbrace{\|\overline{\mathbf{e}}_{\mathsf{LEnc}}\|_\infty}_{\leq \max \overline{\chi}_{\mathsf{LEnc}}} \\ &\leq \Delta/2 - B_{\mathsf{LEnc}} - (B_{\mathsf{LEnc}} - \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}) \,,\end{aligned}$$

and therefore Equation 2 is fulfilled.

Efficiency. Revisiting Equation 3, we see that (for fixed i, j, $d_{i,j}$) rejection independently happens with probability $P := \frac{B_{\mathsf{LEnc}}+B_{\mathsf{LHE}}+\max \overline{\chi}_{\mathsf{LEnc}}}{\Delta/2}$. We can use this together with Chernoff in order to bound the total number of "rejections" $R := \sum_{i\in[w'],j\in[n]} d_{i,j}$, i.e., the number of times where Equation 3 does not hold during encryption, by $R \leq O(P \cdot w'n + \lambda^2)$ with probability $1 - \mathsf{negl}(\lambda)$. (Whenever the average number of rejections is $P \cdot w'n > \lambda$, then with overwhelming probability, the total number of rejections will be $\leq O(P \cdot w'n)$. Whenever the average number of rejections is $P \cdot w'n \leq \lambda$, then with overwhelming probability, the total number of rejections will be $\leq O(\lambda^2)$.)

Assuming $P \leq \frac{1}{2}$, the size of the list $(d_{i,j})_{i\in[w'],j\in[n]}$ (and therefore also the size of ciphertext $|\mathsf{ct}_2|$) is roughly $2w'n + \mathrm{poly}(\lambda)$. Alternatively, for small P it is more beneficial to express $(d_{i,j})_{i\in[w'],j\in[n]}$ within $R \cdot \log(w'n)$ bits (by listing all pairs (i,j), potentially including duplicates, for which a rejection took place). Hence, assuming $w \geq \Omega(n)$ and therefore $w'n \leq O(w)$, the new size of $\mathsf{Sel.ct}_2$ is now (with overwhelming probability)

$$|\mathsf{seed}| + |(d_{i,j})_{i\in[w'],j\in[n]}| \leq \lambda + R \cdot \log(w'n) \leq O(P \cdot w \log w) + \mathrm{poly}(\lambda)$$

instead of $O(w \log q)$.

Furthermore, whenever $B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} < \Delta/2^\lambda$, rejection happens only with probability $P \leq O(\frac{1}{2^\lambda})$. Thus, we can choose ct_2 to be *completely empty* (as all $d_{i,j}$ will be 0 with overwhelming probability, and we may insert the λ-size seed as a new component into $\mathsf{sk}_{\mathbf{y}}$) while still maintaining correctness with probability $1 - \frac{\mathrm{poly}(\lambda)}{2^\lambda}$.

Security. Intuitively, security follows from the fact that rejection happens with *public* probability, and therefore each $d_{i,j}$ can be simulated easily. However, we additionally need to ensure that all $H(\mathsf{seed}, i, j, d')$ for $d' < d_{i,j}$ are programmed in such a way that Equation 3 actually fails. We give a formal proof in Appendix D.1.

For our construction above, we can e.g. get the following parameter setting when considering a message space $\mathcal{M} = \mathbb{Z}_p^\ell$ with $\log |\mathcal{M}| = \ell \cdot \log p \geq \lambda$, proven in Appendix D.2.

Theorem 4 (*Random* Batch-Select). *Consider the same assumptions and parameters as in Theorem 3, except that $q \geq n^{25}$ and $p = \Theta(q^\varepsilon)$ for some $\varepsilon > 0$. Then, there exists a $\sqrt{q}$-times 2^λ-simulation secure* random *batch-select scheme with message space $\mathbb{Z}_p^\ell$ of size $|\mathcal{M}| \geq 2^\lambda$ in the ROM with the same efficiencies as described in Theorem 3, except that the ciphertext ct_2 has only size $O(\frac{w \cdot \lambda}{q^{1/2-\varepsilon} \cdot \log q}) + \mathrm{poly}(\lambda)$.*

Furthermore, when $q > 2^{2(1+\varepsilon')\lambda}$ while $p = \Theta(q^{\varepsilon'\lambda})$ for some $\varepsilon' > 0$, then there is a batch-select scheme with the same properties, except that the size of ct_2 is 0.

5 Application: Preprocessing Garbling

In this section, we apply our construction of batch-select towards the notion of *preprocessing garbling* with succinct function-dependent garbling. We first describe the model in Section 5.1 and then our desired primitive in Section 5.2. Afterwards, we define some existing building blocks that we require (apart from batch-select) in Section 5.3, and give the final construction in Section 5.4.

5.1 Computational Model

A *preprocessing garbling scheme* will handle computations specified by a *universal* function and a *function* description. For example, a universal function may specify parameters of a Boolean

circuit such as input and output lengths as well as circuit size, while the function description may specify the gate types and how they are connected.

We formalize this model of computation as a family $\mathcal{U} = \{\mathcal{U}_{\ell_x,\ell_y}\}_{\ell_x,\ell_y\in\mathbb{N}}$ of classes of *universal* functions $U \in \mathcal{U}_{\ell_x,\ell_y}$ with the format

$$U : \mathcal{F}_U \times \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y} ,$$

where $f \in \mathcal{F}_U$ is the *function* description. The two components U, f together with an input bit string $\mathbf{x} \in \{0,1\}^{\ell_x}$ determine the output $\mathbf{y} = U(f, \mathbf{x})$. Our preprocessing garbling construction will take as input the universal function U in as a Boolean circuit, and $f \in \mathcal{F}_U$, a string.

In the case where U is a universal function capable of handling any function f that is specified by a binary circuit, we may use the following result from prior work, which shows that there exists efficient universal circuits.

Lemma 7 ([ZYZL19]). *There is a Boolean circuit C_U of size $17.75n \log n$ (with $4.5n \log n$ AND gates) that takes a function description $f : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$ (represented by a Boolean circuit with at most n gates) and an input $\mathbf{x} \in \{0,1\}^{\ell_x}$, and outputs $f(\mathbf{x})$.*

5.2 Definition

In the following definition for *preprocessing garbling*, the algorithms RDGen, $\mathsf{InputKeyGen}$, and $\mathsf{GarbleU}$ together form the *function-independent offline* phase that only depends on the universal function U. The *function-dependent offline* phase corresponds to $\mathsf{GarbleFunc}$, which then takes the function description f. The ultimate goal, as achieved by our instantiation, is that the output of $\mathsf{GarbleFunc}$ is *succinct*, i.e., its length is independent of that of the function description f.

Definition 14 (Preprocessing Garbling). *Let $\mathcal{U} = \{\mathcal{U}_{\ell_x,\ell_y}\}$ be a class of computation, where each universal function $U \in \mathcal{U}_{\ell_x,\ell_y}$ has a signature $U : \mathcal{F}_U \times \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$. A preprocessing garbling scheme (with reusability) for $\mathcal{U}$ consists of five efficient algorithms, and is associated with a key space $\mathcal{K}(\lambda)$, which is an abelian group with size $|\mathcal{K}(\lambda)| \geq 2^\lambda$.*

- $\mathsf{RDGen}(U)$ *takes the universal function U and outputs the reusable part $\widehat{U}_{\mathsf{rd}}$ of the function-independent garbling, together with a reusable state $\mathsf{st}_{\mathsf{rd}}$.*
- $\mathsf{InputKeyGen}(1^\lambda, 1^{\ell_x})$ *takes an input length ℓ_x, and returns input keys $\mathbf{K} \in \mathcal{K}^{\ell_x \times 2}$.*
- $\mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K})$ *takes the reusable state $\mathsf{st}_{\mathsf{rd}}$, and the input keys $\mathbf{K}$. It outputs the non-reusable part $\widehat{U}$ of the function-independent garbling, and state st.*
- $\mathsf{GarbleFunc}(\mathsf{st}, f \in \mathcal{F}_U)$ *takes the state st, a function description f, and outputs a hint, i.e., the function-dependent garbling. We refer to $(\widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint})$ as the garbling of (U, f).*
- $\mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{k}_{\mathbf{x}} \in \mathcal{K}^{\ell_x})$ *takes a function description f, the garbling $(\widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint})$, and input labels $\mathbf{k}_{\mathbf{x}}$, and outputs the computation result $\mathbf{y} \in \{0,1\}^{\ell_y}$.*

Correctness. *The scheme is correct if for all $\lambda, \ell_x, \ell_y \in \mathbb{N}$, universal functions $U \in \mathcal{U}_{\ell_x,\ell_y}$ and function descriptions $f \in \mathcal{F}_U$, input keys $\mathbf{K} \in \mathcal{K}^{\ell_x \times 2}$, and inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$, the following holds:*

$$\Pr\left[\begin{array}{l}\mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{K}[\mathbf{x}]) \\ = U(f, \mathbf{x})\end{array} \;\middle|\; \begin{array}{r}(\widehat{U}_{\mathsf{rd}}, \mathsf{st}_{\mathsf{rd}}) \leftarrow \mathsf{RDGen}(U) \\ (\widehat{U}, \mathsf{st}) \leftarrow \mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K}) \\ \mathsf{hint} \leftarrow \mathsf{GarbleFunc}(\mathsf{st}, f)\end{array}\right] = 1.$$

Definition 15 (T-times Input Privacy). *The scheme fulfills T-times input privacy if there exists an efficient simulator $\mathsf{Sim}_{\mathsf{Priv}}$ s.t. for any efficient adversary $\mathcal{A}$,*

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

where the game $\mathsf{Exp}^{\mathcal{A},b}_{\mathsf{Priv}}(\lambda)$ is defined below.

1. *$\mathcal{A}(1^\lambda)$ selects $1^{\ell_\mathsf{x}}, 1^{\ell_\mathsf{y}}$, a universal function $U\in\mathcal{U}_{\ell_\mathsf{x},\ell_\mathsf{y}}$, and T function descriptions $f^{(t)}\in\mathcal{F}_U$, and inputs $\mathbf{x}^{(t)}\in\{0,1\}^{\ell_\mathsf{x}}$, for $t\in[T]$.*
2. *The game sends to $\mathcal{A}$ the reusable garbling $\widehat{U}_{\mathsf{rd}}$ and the T individual garblings with their input keys $\{\widehat{U}^{(t)},\mathsf{hint}^{(t)},\mathbf{k}^{(t)}_\mathbf{x}\}_{t\in[T]}$, computed as follows.*
   - *If $b=0$, first run $(\widehat{U}_{\mathsf{rd}},\mathsf{st}_{\mathsf{rd}})\leftarrow\mathsf{RDGen}(U)$. Then for $t\in[T]$ run*

$$\mathbf{K}^{(t)}\leftarrow\mathsf{InputKeyGen}(1^\lambda,1^{\ell_\mathsf{x}}),$$
$$(\widehat{U}^{(t)},\mathsf{st}^{(t)})\leftarrow\mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}},\mathbf{K}^{(t)}),\qquad \mathsf{hint}^{(t)}\leftarrow\mathsf{GarbleFunc}(\mathsf{st}^{(t)},f^{(t)})$$

     Finally, set $\mathbf{k}^{(t)}_\mathbf{x}=\mathbf{K}^{(t)}[\mathbf{x}]$.
   - *If $b=1$, run $(\widehat{U}_{\mathsf{rd}},\{\widehat{U}^{(t)},\mathsf{hint}^{(t)},\mathbf{k}^{(t)}_\mathbf{x}\})\leftarrow\mathsf{Sim}_{\mathsf{Priv}}(U,\{f^{(t)},U(f^{(t)},\mathbf{x}^{(t)})\}_{t\in[T]})$.*
3. *$\mathcal{A}$ outputs a bit b', which is also the output of the game.*

5.3 Ingredients

Standard Model Garbling. Traditional garbling (i.e., the standard model in which there is no separation between universal function and function description) may be seen as a special case of preprocessing garbling. It will be required for our construction of preprocessing garbling.

Definition 16 (Standard Model Garbling). *Let $\mathcal{C}=\{\mathcal{C}_{\ell_\mathsf{x},\ell_\mathsf{y}}\}_{\ell_\mathsf{x},\ell_\mathsf{y}\in\mathbb{N}}$ be the family of functions that do not take any function description, i.e., $\mathcal{C}_{\ell_\mathsf{x},\ell_\mathsf{y}}$ consists of all two-input functions $C:\emptyset\times\{0,1\}^{\ell_\mathsf{x}}\to\{0,1\}^{\ell_\mathsf{y}}$, specified by their Boolean circuit representation.*

For this class, a corresponding garbling scheme (which we denote by SG, "standard garbling") without support for preprocessing or reusability has the following simplified syntax, where RDGen and $\mathsf{GarbleFunc}$ are not needed anymore, and $\mathsf{GarbleU}$ is renamed to Garble.

- $\mathsf{SG.InputKeyGen}(1^\lambda,1^{\ell_\mathsf{x}})$ *takes an input length ℓ_x, and returns input keys $\mathbf{K}\in\mathcal{K}^{\ell_\mathsf{x}\times 2}$.*
- $\mathsf{SG.Garble}(C,\mathbf{K})$ *takes the circuit C and the input keys $\mathbf{K}$, and outputs the garbling $\widehat{C}$.*
- $\mathsf{SG.Eval}(C,\widehat{C},\mathbf{k}_\mathbf{x}\in\mathcal{K}^{\ell_\mathsf{x}})$ *takes the circuit C, its garbling $\widehat{C}$ and input labels $\mathbf{k}_\mathbf{x}$, and outputs the computation result $\mathbf{y}\in\{0,1\}^{\ell_\mathsf{y}}$.*

Further, we require that $\mathsf{SG.InputKeyGen}(1^\lambda,1^{\ell_\mathsf{x}})$ samples each of the ℓ_x input key pairs individually, i.e., its implementation only consists of running $(\mathbf{K}[0,i],\mathbf{K}[1,i])\leftarrow\mathsf{InputKeyGen}'(1^\lambda)$ for some algorithm $\mathsf{InputKeyGen}'$.

Because we do not require reusability for the standard garbling scheme SG, we use the term input privacy *to denote* 1-time input privacy *(Definition 15).*

Garbling schemes in the standard model as above are known to exist, with the current state-of-the-art producing a garbling whose size is 1.5λ times the number of AND gates.

Lemma 8 ([RR21]). *There is a standard model garbling scheme SG (with global key offsets), with garblings of size $|\widehat{C}|=(1.5\lambda+10)n$ bits, where n denotes the number of AND gates in the circuit C.*

Correlation-robust hash functions In our construction of preprocessing garbling, we will need to translate the output of a batch-select scheme (which is in $\mathcal{M}$, or concretely $\mathcal{M} = \mathbb{Z}_p^\ell$ for our instantiation in Section 4.3) into keys in $\mathcal{K}$ for the input wires of a circuit C_U garbled using a standard garbling scheme SG.

We do so by hiding a key in $\mathcal{K}$ using a hash function $H : \mathcal{M} \to \mathcal{K}$ applied to the corresponding batch-select output that is in $\mathcal{M}$. Furthermore, whenever our garbling scheme is reused ($T > 1$), we will need to ensure that H is correlation-robust in the following sense.

Definition 17 (Correlation-robust hash function). *A hash function $H : \mathcal{M} \to \mathcal{K}$ (where $\mathcal{M}$ and $\mathcal{K}$ are parameterized by the security parameter λ) is* correlation-robust*, if for any polynomial $T(\lambda)$ and efficient adversaries $\mathcal{A}$, the following holds (where we suppress λ):*

$$\Big|\Pr[\mathcal{A}(\{\mathbf{l}^{(t)}, H(\mathbf{l}^{(t)} - \Delta), H(\mathbf{l}^{(t)} + \Delta)\}_{t\in[T]}) = 1 \mid \Delta, \mathbf{l}^{(1)}, \ldots, \mathbf{l}^{(T)} \leftarrow\!\$\, \mathcal{M}] - \Pr\left[\mathcal{A}(\{\mathbf{l}^{(t)}, \mathbf{u}_-^{(t)}, \mathbf{u}_+^{(t)}\}_{t\in[T]}) = 1 \;\middle|\; \begin{array}{l} \mathbf{l}^{(1)}, \ldots, \mathbf{l}^{(T)} \leftarrow\!\$\, \mathcal{M}, \\ \mathbf{u}_-^{(1)}, \ldots, \mathbf{u}_-^{(T)}, \mathbf{u}_+^{(1)}, \ldots, \mathbf{u}_+^{(T)} \leftarrow\!\$\, \mathcal{K} \end{array}\right]\Big| \leq \mathsf{negl}(\lambda)$$

Due to $|\mathcal{M}| \geq 2^\lambda$, the correlation-robust hash function H can be instantiated using a random oracle. Also note that it is a *weaker notion* of the type of circular correlation-robustness that is already utilized by many existing garbling schemes involving e.g. the free-XOR optimization [KS08].

5.4 Construction

We will construct a preprocessing garbling scheme for any computation class $\mathcal{U}$, s.t. for any universal function $U \in \mathcal{U}_{\ell_\mathsf{x},\ell_\mathsf{y}}$, there is a universal circuit C_U that takes a function description $f \in \mathcal{F}_U$ whose bit-length is bounded by some s_U, an input $\mathbf{x} \in \{0,1\}^{\ell_\mathsf{x}}$, and outputs $U(f, \mathbf{x})$.

For our construction, we assume a standard garbling scheme SG with key space $\mathcal{K}$ and a batch-select with message space $\mathcal{M} = \mathbb{Z}_p^\ell$. Further, we assume a correlation-robust hash function $H : \mathcal{M} \to \mathcal{K}$.

Construction 4 (Preprocessing Garbling).

$\underline{\mathsf{RDGen}(U) \to \widehat{U}_\mathsf{rd}, \mathsf{st}_\mathsf{rd}}$**:** *(This algorithm creates reusable data, which can be used again for subsequent garbling sessions.)*

First setup the batch-select Sel scheme with dimension s_U, where s_U is the description length of functions in $\mathcal{F}_U$:

$$\mathsf{Sel.pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}).$$

Next, sample the first message vector $\mathbf{l}_1$, and encrypt it using $\mathsf{Sel.Enc}_1$:

$$\mathbf{l}_1 \leftarrow\!\$\, \mathcal{M}^{s_U}, \quad (\mathsf{Sel.ct}_1, \mathsf{Sel.st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1).$$

Output the reusable part of the garbling $\widehat{U}_\mathsf{rd} = (\mathsf{Sel.pp}, \mathsf{Sel.ct}_1)$, and the reusable state $\mathsf{st}_\mathsf{rd} = (\mathsf{Sel.pp}, \mathsf{Sel.st}_1, \mathbf{l}_1)$.

$\underline{\mathsf{InputKeyGen}(1^\lambda, 1^{\ell_\mathsf{x}}) \to \mathbf{K}}$**:** Output input keys $\mathbf{K} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{\ell_\mathsf{x}})$.

$\underline{\mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K}) \to \widehat{U}, \mathsf{st}}$**:** First, sample the input keys $\mathbf{K}^{\mathsf{Fn}}$ for the input wires of the universal circuit C_U that correspond to the function description, and use this to garble the universal circuit C_U:

$$\mathbf{K}^{\mathsf{Fn}} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{s_U}), \quad \widehat{C}_U \leftarrow \mathsf{SG.Garble}(C_U, (\mathbf{K}^{\mathsf{Fn}}, \mathbf{K})) \ .$$

Next, sample the second message vector $\mathbf{l}_2$, and encrypt it using $\mathsf{Sel.Enc}_2$:

$$\mathbf{l}_2 \leftarrow\!\$\ \mathcal{M}^{s_U}, \quad (\mathsf{Sel.ct}_2, \mathsf{Sel.st}_2) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2).$$

Then, for each $i \in [s_U]$, produce the ciphertexts $\mathsf{ct}'_{0,i}$ and $\mathsf{ct}'_{1,i}$ that the evaluator will use to translate the batch-select output $\mathbf{f}[i] \cdot \mathbf{l}_1[i] + \mathbf{l}_2[i]$ into the correct key $\mathbf{K}^{\mathsf{Fn}}[i, \mathbf{f}[i]]$:

$$\begin{aligned} \mathsf{ct}'_{0,i} &:= H(\mathbf{l}_2[i]) + \mathbf{K}^{\mathsf{Fn}}[i, 0] \\ \mathsf{ct}'_{1,i} &:= H(\mathbf{l}_1[i] + \mathbf{l}_2[i]) + \mathbf{K}^{\mathsf{Fn}}[i, 1] \end{aligned}$$

Output function-independent garbling $\widehat{U} = (\widehat{C}_U, \mathsf{Sel.ct}_2, \{\mathsf{ct}'_{0,i}, \mathsf{ct}'_{1,i}\})$, and the state $\mathsf{st} = (\mathsf{Sel.pp}, \mathsf{Sel.st}_1, \mathsf{Sel.st}_2)$.

$\underline{\mathsf{GarbleFunc}(\mathsf{st}, f) \to \mathsf{hint}}$**:** Let $\mathbf{f} \in \{0,1\}^{s_U}$ denote the bit representation of f. Output a batch-select secret key for selecting $\mathbf{K}^{\mathsf{Fn}}[\mathbf{f}]$:

$$\mathsf{hint} = \mathsf{sk}_{\mathbf{f}} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{Sel.st}_1, \mathsf{Sel.st}_2, \mathbf{f}).$$

$\underline{\mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{k}_{\mathbf{x}}) \to \mathbf{y}}$**:** Parse the function-independent garbling as $\widehat{U}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \mathsf{Sel.ct}_1)$ and $\widehat{U} = (\widehat{C}_U, \mathsf{Sel.ct}_2, \{ct'_{0,i}, \mathsf{ct}'_{1,i}\})$. Let $\mathbf{f} \in \{0,1\}^{s_U}$ denote the bit representation of f. First decrypt the batch-select ciphertexts to recover the messages $\mathbf{l}_{\mathsf{res}}$ selected by $\mathbf{f}$:

$$\begin{aligned} & \mathbf{l}_{\mathsf{res}} \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_{\mathbf{f}}, \mathsf{Sel.ct}_1, \mathsf{Sel.ct}_2, \mathbf{f}), \\ // \text{ s.t.} \quad & \mathbf{l}_{\mathsf{res}} = \mathbf{l}_1 \odot \mathbf{f} + \mathbf{l}_2 \end{aligned}$$

Then, translate these messages into the appropriate input keys $\mathbf{k}^{\mathsf{Fn}}_{\mathbf{f}}$ by computing, for each $i \in [s_U]$:

$$\begin{aligned} & \mathbf{k}^{\mathsf{Fn}}_{\mathbf{f}}[i] := \mathsf{ct}'_{\mathbf{f}[i],i} - H(\mathbf{l}_{\mathsf{res}}[i]) \ , \\ // \text{ s.t.} \quad & \mathbf{k}^{\mathsf{Fn}}_{\mathbf{f}}[i] = \mathbf{K}^{\mathsf{Fn}}[i, \mathbf{f}[i]] \ . \end{aligned}$$

Then evaluate the garbled universal circuit as $\mathbf{y} \leftarrow \mathsf{SG.Eval}(\widehat{C}_U, (\mathbf{k}^{\mathsf{Fn}}_{\mathbf{f}}, \mathbf{k}_{\mathbf{x}}))$, and output $\mathbf{y}$.

Correctness and security. Correctness follows straightforwardly from that of the standard model garbling scheme SG and that of the batch-select scheme Sel. We state security formally below, and defer the proof to Appendix E.

Lemma 9. *Assuming the standard garbling scheme* SG *satisfies input privacy, the batch-select scheme* Sel *satisfies* T*-times simulation security (Definition 6), and the hash function* H *is correlation-robust (Definition 17), the preprocessing garbling scheme in Construction 4 fulfills* T*-times input privacy (Definition 15).*

Efficiency. With our instantiation of batch-select, the function-dependent phase (consisting only of sending $\mathsf{sk_f}$) is succinct: The $\mathsf{hint} = \mathsf{sk_f}$ consists of a single ring element in $\mathcal{R}_q$. The cost of $\widehat{U}$ essentially corresponds to the cost for garbling a universal circuit $\widehat{C}_U$ (e.g. $|\widehat{C}_U| = \lambda \cdot n \log n$ to support arbitrary Boolean circuits with at most n gates; see Lemmas 7 and 8). The *reusable* part $\widehat{U}_{\mathsf{rd}}$ may have size $O(\lambda \cdot n \log^2 n)$, but one factor of $\log n$ can be avoided by using our optimized batch-select constructions (i.e., reusability or *weak* batch-select).

Note regarding Key Translation. The reason for including key translation ciphertexts $\mathsf{ct}'_{0,i}, \mathsf{ct}'_{1,i}$ is the potential mismatch between the batch-select message space $\mathcal{M} = \mathbb{Z}_p^\ell$, and the SG key space $\mathcal{K}$ (which is typically of the form $\{0,1\}^\lambda$). If those two were equal (many existing garbling schemes could theoretically work with any sufficiently large key space, potentially at the cost of not supporting free-XOR anymore), then we may omit key translation ciphertexts. Orthogonally, if no reusability is needed ($T = 1$), we could also directly choose messages $\mathbf{l}_1 := \phi(\mathbf{K}^{\mathsf{Fn}}[1] - \mathbf{K}^{\mathsf{Fn}}[0])$ and $\mathbf{l}_2 := \phi(\mathbf{K}^{\mathsf{Fn}}[0])$ for any injective embedding $\phi : \mathcal{K} \to \mathcal{M}$, avoiding key translation completely despite $\mathcal{K} \neq \mathcal{M}$.

6 Evaluation

To demonstrate concrete efficiency, we have implemented our batch-select scheme (Construction 3). Here, our primary focus is *computational efficiency* instead of optimal (near-0) offline communication cost that would requires exponential modulus (as described e.g. in Theorem 3). Nevertheless, our (amortized) offline cost will be just ≈ 6 bits per 128-bit value transferred to a receiver.

Parameter Setting. We consider the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{4096} + 1)$ of degree $n = 4096$, and as required by our batch-select scheme, we choose the modulus $q = p \cdot \Delta$ as a product of two primes p, Δ. In order to be able to perform efficient computations, we choose the moduli p and Δ as two different NTT-friendly primes (i.e., $p \equiv \Delta \equiv 1 \bmod 8192$). Using the Chinese Remainder Theorem, this allows us to represent an element of the ring $\mathcal{R}_q$ as one $\mathcal{R}_p$-element and one $\mathcal{R}_\Delta$-element. Multiplication and addition over $\mathcal{R}_q$ can then be performed separately over $\mathcal{R}_p$ and $\mathcal{R}_\Delta$. We choose p as a 50-bit prime (then, three elements of the plaintext space $\mathbb{Z}_p$ will be enough to store one 128-bit key for our garbling application), and Δ as a 59-bit prime (this is the maximum bitlength supported by the SEAL framework [SEA23] we build upon).

For concreteness, suppose we apply Sel to vectors of $w' = 512$ ring elements, i.e., vectors of $w = w' \cdot n = 512 \cdot n = 2^{21}$ elements in the message space $\mathcal{M} = \mathbb{Z}_p$. This means that $\lfloor \frac{2^{21}}{3} \rfloor = 699\,050$ elements in $\mathbb{Z}_p^3$ are supported. (Recall that we are interested in $\mathbb{Z}_p^3$, because its size is larger than 2^λ, and hence we can encrypt 128-bit input labels in it.)

We are now aiming to provide $\lambda = 128$-bit security. As recommended by the homomorphic encryption standard [ACC+19], given the degree-4096 ring $\mathcal{R}_q$ (with $\log q \leq 111$) we assume that the Ring-LWE assumption $\mathsf{LWE}_{\mathcal{R},\infty,q,\chi^*}$ holds with Gaussian noise of standard deviation $\sigma^* = 8/\sqrt{2\pi} \approx 3.2$, i.e., with error distribution $\chi^* = \mathcal{D}_{\mathcal{R},s^*}$ of parameter $s^* = 8$.

Note that the security of LEnc and LHE (as invoked by our T-reusable batch-select scheme) relies on the elLWE assumption with two different types of leakage sets: $\mathcal{L}_{T,2m}(g)$ and $\mathcal{L}_{1,1}^{\times w'}(g)$. To ensure that this follows from our assumption $\mathsf{LWE}_{\mathcal{R},\infty,q,\chi^*}$, we combine Theorem 2 with Lemmas 3 and 4. It guarantees that it suffices to choose parameters s and $\overline{s}$ as follows, where s is used for distributions $\chi = \mathcal{D}_{\mathcal{R},s}$ inside both LHE and LEnc, and $\overline{s}$ is used for both distributions

$\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ (used to hide LEnc noise leakage) and $\overline{\chi} = \mathcal{D}_{\mathcal{R},\overline{s}}$ (used inside LHE):

$$s = 2s^* + 1 + \eta_\epsilon(\mathcal{R}) \quad \text{and} \quad \overline{s} = (s+1) \cdot g \cdot n \cdot \sqrt{2m \cdot T}\ .$$

In the above, there are some tunable parameters: m (used inside our LHE and LEnc constructions, where the gadget vector is determined through its *base* g and *dimension* m with $g^m < q$) affects the ciphertext size and therefore the amount of communication, and T denotes how often ct_1 may be reused. We can freely choose these parameters, as long as correctness is guaranteed, i.e., $\Delta \geq 2\sqrt{\lambda}(g \cdot m \cdot d \cdot s \cdot (\lceil \log_2 w' \rceil + 1) + 2\overline{s})$ still holds given the choices of s and $\overline{s}$ above. For our evaluation, we choose

$$T = 2^{15} = 32\,768 \quad \text{and} \quad m = 4\ ,$$

and therefore also $g = 2^{28}$.

Communication. A single ring element can be represented with 56 KB (4096 coefficients with 109 bits each). Therefore, our basic batch-select construction (without RO-optimization) achieves the following efficiencies (where $m = 4$ is the decomposition multiplier in both our LEnc and LHE constructions):

- Public parameters pp: $w' + 2m$ ring elements (≈ 29 MB)
- Reusable ciphertext ct_1: $m \cdot w' + 2m \cdot w' \lceil \log_2 w' \rceil$ ring elements (≈ 2.2 GB, or when amortized over $T = 2^{15}$ iterations: 67 KB)
- Non-reusable ciphertext ct_2: w' ring elements (≈ 29 MB)
- Key $\mathsf{sk}_\mathbf{y}$: 1 ring element (≈ 56 KB)

The following two optimizations regarding *communication size* may be applied:

- *Weak* laconic encryption (see Construction 5) would reduce the size of ct_1 to $m \cdot w' + 4m \cdot w'$ ring elements (≈ 571 MB instead of ≈ 2.2 GB). However, this would simultaneously require the noise $\overline{s}$ to increase by another factor of $\sqrt{w'}$; hence reducing the maximum number T of uses for ct_1 by a factor of $w' = 512$ (i.e., $T \leq 2^6$).
- The RO optimization (see Section 4.4) allows reducing the size of ct_2: Instead of sending w' ring elements, it suffices to send a λ-bit seed, plus $2w'n = 2w$ rejection sampling-bits. This reduces the size of ct_2 to just ≈ 524 KB in our setting. Combining this with amortization over $T = 2^{15}$ iterations, the *total communication cost* per iteration, $|\mathsf{ct}_1|/T + |\mathsf{ct}_2|$, will be only 591 KB (i.e., only ≈ 6.4 bits per transferred element in $\mathbb{Z}_p^3$).

Computation optimizations. Note that naive multiplication of two polynomials stored in coefficient form is very expensive: first, both polynomials need to be converted to NTT form, then they are multiplied component-wise, and then another inverse-NTT needs to be performed. To optimize this, our implementation keeps all polynomials in NTT form whenever possible. Then, each multiplication requires only one component-wise multiplication.

NTT transformations are now only required when generating or removing noise, or when converting the LEnc database from $\mathbf{a} \in \mathcal{R}_p^{w'}$ to the larger modulus $\mathcal{R}_q^{w'}$. In addition—this is the dominating use of NTT transformations in the input-dependent phase consisting of $\mathsf{Sel.KeyGen}$ and $\mathsf{Sel.Dec}$—computing the hash-tree inside $\mathsf{LEnc.Digest}$ requires $4mw'$ forward-NTT and $2w'$ inverse-NTT.

Hardware setup. We implemented our batch-select scheme on top of the SEAL library [SEA23] (which allows us to utilize existing implementations of ring operations), and tested it on a server

	Reusable		One-time	Input-dependent	
	Setup	Enc_1	Enc_2	KeyGen	Dec
Conjectured time	$0.13s$	$14.93s$	$0.17s$	$0.44s$	$0.71s$
Time	$0.13s$	$16.65s$	$0.25s$	$0.63s$	$1.24s$
Time / msg	$0.18\mu s$	$23.83\mu s$	$0.35\mu s$	$0.90\mu s$	$1.77\mu s$
Time ($T = 2^{15}$)	$3.8\mu s$	$0.0005s$	"	"	"
	pp	ct_1	ct_2	$\mathsf{sk}_\mathbf{y}$	n/a
Size	29MB	2.2GB	29MB	56KB	
Size / msg	42 byte	3.1KB	41 byte	0.08 byte	
Size ($T = 2^{15}$)	885 byte	67KB	"	"	
Size / msg ($T = 2^{15}$)	< 0.01 byte	0.1 byte	"	"	
Size w/ RO opt.			524 KB		
Size / msg w/ RO opt.			0.75 byte		

Table 4: This table shows the amount of time spent within each of the algorithms of our batch-select scheme, and the size of their outputs. All *time* rows (except the first one) show the actual times required by our implementation, where *per msg* is the time required on average by each of the $w = 699\,050$ messages (i.e., input labels) encrypted by the batch-select scheme, and $T = 2^{15}$ indicates that we amortize the reusable parts across 2^{15} iterations. The first row shows the computation time that we conjecture for a CPU that has AVX512-IFMA52 instructions. These numbers are estimated using the benchmarks from [BKS+21], where one multiplication requires $1.08\mu s$, one forward NTT requires $5.81\mu s$, and one inverse NTT requires $5.72\mu s$.

	Add. (per msg)	Mult. (per msg)	NTT (per msg)	CRT (per msg)
Setup	0	0	0	0
Enc_1	$0.46s$ ($0.66\mu s$)	$0.79s$ ($1.13\mu s$)	$1.30s$ ($1.86\mu s$)	0
	118 784 (0.17)	77 824 (0.11)	77 824 (0.11)	
Enc_2	$0.01s$ ($0.01\mu s$)	$0.01s$ ($0.02\mu s$)	$0.03s$ ($0.04\mu s$)	0
	3072 (0.004)	1024 (0.001)	2024 (0.003)	
KeyGen	$0.01s$ ($0.01\mu s$)	$0.05s$ ($0.07\mu s$)	$0.17s$ ($0.24\mu s$)	$0.28s$ ($0.40\mu s$)
	8184 (0.01)	8184 (0.01)	11 254 (0.02)	4092 (0.01)
Dec	$0.09s$ ($0.125\mu s$)	$0.56s$ ($0.801\mu s$)	$0.20s$ ($0.288\mu s$)	$0.28s$ ($0.396\mu s$)
	97 776 (0.14)	87 024 (0.12)	12 278 (0.02)	4092 (0.01)

Table 5: For each function of our batch-select scheme, this table shows the computation time and the number of invocations required by our implementation (in parentheses the time and number required on average for each of the $w = 699\,050$ messages), for the most time-intensive components: (1) the number of polynomial additions (also including subtractions), (2) the number of component-wise multiplications in NTT form, (3) the number of NTT transformations (including both forward and inverse transformations), and (4) the number of CRT decompositions (the number of CRT compositions, ommitted from the table, is lower by a factor of m). In Enc_1, the majority of time is spent on generating noise polynomials.

with an Intel Xeon Platinum 8160M Processor, 2.10GHz. The implementation utilizes only a single core.

We achieve an additional speedup using the Intel HEXL framework [BKS+21], which utilizes the Intel AVX512 instruction set. However, the server we tested on only supports AVX512-DQ instructions, and therefore we can expect that a CPU with the AVX512-IFMA52 instruction set would be able to improve the computation time of NTT operations by another factor of 3.

Evaluation results. Table 4 displays the amount of computation time and the communication size on the discussed parameters (while our implementation does not directly support the RO-optimization, we do not expect the running time of Enc_2 to increase significantly when it generates its output from RO). In Table 5, we further split the computation time to demonstrate which operations are causing the highest cost.

Putting these results into context, suppose the batch-select scheme is used for transmitting input keys (of bitlength $\lambda = 128$) for a garbled circuit. Naively sending input labels without batch-select requires sending more than 11 MB in the online phase. On the other hand, batch-select allows a tradeoff: only 56 KB need to be sent (once the input is known), in exchange for $1.87s$ of computation time. Thus, our scheme could outperform the naive baseline whenever the bandwidth is limited by about 45 Mbps.

Also note that our scheme allows for parallelization (because most computation happens independently for each of the $w' = 512$ components). For example, when using 8 cores instead of 1 core, computation time in the input-dependent phase could be as small as $0.24s$, outperforming the naive baseline whenever the bandwidth is limited by 350 Mbps.

Acknowledgments. The authors were supported by NSF grant CNS-2026774, and a Simons Collaboration on the Theory of Algorithmic Fairness.

References

ABI+23. Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1553–1566. ACM Press, June 2023.

ACC+19. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption standard. Cryptology ePrint Archive, Report 2019/939, 2019.

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.

AIKW13. Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Berlin, Heidelberg, August 2013.

AJS17. Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for Turing machines: Constant overhead and amortization. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 252–279. Springer, Cham, August 2017.

AL21. Martin R. Albrecht and Russell W. F. Lai. Subtractive sets over cyclotomic rings - limits of Schnorr-like arguments over lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 519–548, Virtual Event, August 2021. Springer, Cham.

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.

BCG+20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Cham, August 2020.

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.

BDGM19. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 407–437. Springer, Cham, December 2019.

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.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

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. Cryptology ePrint Archive, Report 2021/420, 2021.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CKKZ12. Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Berlin, Heidelberg, March 2012.

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.

CRR21. Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 502–534, Virtual Event, August 2021. Springer, Cham.

CWYY23. Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 35–67. Springer, Cham, April 2023.

DIJL23. Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 315–348. Springer, 2023.

DILO22. Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87. Springer, Cham, August 2022.

DKL+23. Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 417–446. Springer, Cham, April 2023.

DNPR16. Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael Raskin. On the communication required for unconditionally secure multiplication. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 459–488. Springer, Berlin, Heidelberg, August 2016.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.

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.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

GH19. Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 438–464. Springer, Cham, December 2019.

GHS12. 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.

GKP+13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai 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.

GLNP15. Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 567–578. ACM Press, October 2015.

GOS18. Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 515–544. Springer, Cham, August 2018.

GS18. Sanjam Garg and Akshayaram Srinivasan. Adaptively secure garbling with near optimal online complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 535–565. Springer, Cham, April / May 2018.

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.

HW15. Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

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

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Berlin, Heidelberg, August 2008.

JLL23. Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 479–510. Springer, Cham, April 2023.

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.

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Berlin, Heidelberg, August 2014.

KRRW18. Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, Cham, August 2018.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Berlin, Heidelberg, July 2008.

LP11. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, Berlin, Heidelberg, March 2011.

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

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.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Berlin, Heidelberg, December 2009.

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

RR21. Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Cham.

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

SEA23. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

SV10. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 420–443. Springer, Berlin, Heidelberg, May 2010.

WRK17. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Berlin, Heidelberg, April 2015.

ZYZL19. Shuoyao Zhao, Yu Yu, Jiang Zhang, and Hanlin Liu. Valiant's universal circuits revisited: An overall improvement and a lower bound. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 401–425. Springer, Cham, December 2019.

A Related Works on Sublinear 2PC Protocols and Succinct Garbling

In this section, we summarize the state of sublinear 2PC and succinct garbling, and compare them with our solution in the preprocessing model.

Succinct Garbling In the literature, two types of garbling schemes give drastically different computation vs. communication trade-offs. Yao's original garbled circuits and the successful line of optimization upon it [BMR90, NPS99, KS08, PSSW09, KMR14, GLNP15, ZRE15, RR21] all rely on extremely efficient symmetric-key operations (e.g., a few calls to AES per gate of

the circuit), making communication, rather than computation, the bottleneck. Parties must exchange $O(\lambda)$ bits per gate, resulting in total communication $O(\lambda|C|)$. On the other hand, there are constructions of garbled circuits achieving asymptotically optimal size, i.e., $\mathrm{poly}(\lambda)$, independent of the circuit complexity. Unfortunately, they are computationally expensive, since they rely on heavy machinery, such as, ABE and FHE [GKP+13, HLL23], or (multi-key) Functional Encryption [AJS17, KLW15, BCG+18, JLL23].

Our preprocessing garbled circuits using the batch-select scheme achieves optimal *online* communication $\mathrm{poly}(\lambda)$ and are concretely efficient, by leveraging preprocessing and pre-communication of complexity quasilinear in the circuit size.

We believe that the notion of preprocessing garbling is interesting on its own, as it provides a meaningful middle point between the aforementioned two types of garbling schemes. We note that the quasilinear offline complexity stems from garbling the universal circuits in our construction. However, there might be completely different techniques for constructing preprocessing garbling, perhaps without using the universal circuits. We think this is an interesting technical question.

Sublinear 2PC without Preprocessing In the standard model without preprocessing, the only 2PC protocols for all circuits that achieve communication complexity independent of circuit size rely on powerful tools such as FHE and/or $i\mathcal{O}$. When using FHE [Gen09, BV11, BGV12, GSW13], we can attain protocols where only Alice receives an output, with communication complexity $O_\lambda(|\mathbf{x}_A| + |\mathbf{y}|)$, or $|\mathbf{x}_A| + |\mathbf{y}| + O_\lambda(1)$ if the underlying FHE has a rate-1 property [BDGM19, GH19]. Laconic function evaluation [QWW18] which uses ABE and FHE gives protocols where only Alice receives an output, and the communication complexity is $O_\lambda(|\mathbf{x}_B| + |\mathbf{y}|)$. Combining FHE and $i\mathcal{O}$, the work of [HW15] demonstrated how to eliminate the dependency on output length to achieve communication $|\mathbf{x}_A| + |\mathbf{x}_B| + O_\lambda(1)$ or $O_\lambda(\mathsf{CC}(C))$, where CC represents the communication complexity of computing C without security.

Without FHE or $i\mathcal{O}$, we can still achieve sublinear communication complexity using Homomorphic Secret Sharing (HSS) [BGI16]. An advantage is that HSS can be based on various assumptions that do not imply fully homomorphic encryption, including DDH [BGI16], DCR [FGJS17, OSY21, RS21], assumptions related to class groups of imaginary quadratic fields [ADOS22], and different variants of the Learning Parity with Noise (LPN) assumptions [BCG+19, CM21, DIJL23]. However, current HSS schemes only support low depth computations like NC_1, giving 2PC for low depth computations with communication complexity $O_\lambda(\min(|\mathbf{x}_A|, |\mathbf{x}_B|)+|\mathbf{y}|)$. When extended to circuits, the communication complexity is only slightly sublinear in circuit size, e.g., $|C|/O(\log |C|)$, reducing it by just a logarithmic (or even log-log) factor.

Another drawback of the above techniques is that to achieve malicious security, they need to rely on generic techniques such as communication efficient zero-knowledge protocols, which makes the protocols even more expensive.

In comparison, our 2PC protocols achieve succinct online communication – $\mathrm{poly}(\lambda)$ bits after receiving C, and $|\mathbf{x}_A| + |\mathbf{x}_B| + \mathrm{poly}(\lambda)$ bits after the two parties receiving their inputs – and malicious security, and are concretely efficient. The downside is the offline stage with complexity $\tilde{O}(\lambda|C|)$.

Sublinear 2PC with Preprocessing Recent years have witnessed significant progress in developing practical secure multiparty computation (MPC) protocols within the online-offline model. These protocols, such as those in [IPS08, DPSZ12], utilize the offline stage for conducting computationally expensive cryptographic operations to distribute correlated random coins among the parties. These coins are subsequently used in a fast and information-theoretically secure online computation stage. However, despite these advancements, the communication complexity

of these protocols remains linear in the circuit size. This limitation was highlighted as a major bottleneck and formally investigated in [DNPR16], showing that it is inherent for gate-by-gate protocols.

The exploration of low-communication protocols in the correlated randomness model was initiated in [IKM+13]. In their work, they introduced protocols with communication complexity of $O(|\mathbf{x}_A|+|\mathbf{x}_B|+|\mathbf{y}|)$ but with a requirement for *exponentially* long correlated randomness, referred to as the one-time truth table correlation. They further argued that reducing the amount of correlated random coins from exponential to polynomial for general functions would be challenging and could potentially lead to breakthroughs in long-standing open problems related to private information retrieval. Couteau [Cou19] later extended this approach to achieve polynomially long correlations, albeit with slightly sublinear communication complexity $|C|/O(\log\log|C|)$.

There is a major difference between the preprocessing model considered in these two works and ours: The preprocessing of [IKM+13, Cou19] depends on the function, meaning the correlated randomness is sampled based on the circuit C, only independent of the inputs $\mathbf{x}_A, \mathbf{x}_B$. In contrast, our preprocessing is both function and input independent, providing greater flexibility. On the other hand, the works of [IKM+13, Cou19] aim for information theoretically secure online stage, whereas we settle for computational security. Hence, the models are incomparable. In fact, if we relax to the model to allow both function-dependent preprocessing and computationally secure online stage, it becomes trivial to achieve optimal online communication: Alice can simply send a non-succinct garbled circuit $\widehat{C}$ to Bob in the offline stage, and online they only need communication for obtaining the right input labels.

	Tools	Offline Comm.	Online Comm.	Assumptions
[BDGM19, GH19]	FHE w/ rate-1	/	$\|\mathbf{x}_A\|+\|\mathbf{y}\|+O_\lambda(1)$	LWE w/ circular security
[HW15]	FHE $+\ i\mathcal{O}$	/	$\|\mathbf{x}_A\|+\|\mathbf{x}_B\|+O_\lambda(1)$	LWE w/ circular security +DLIN+LPN +NC^0-PRG
[BGI16, OSY21] [BKS19]	HSS	/	$O(\|C\|/\log\|C\|)$	DDH/DCR/ LWE
[BCG+19, CM21] [DIJL23]	HSS	/	$O(\|C\|/\log\log\|C\|)$	variants of LPN
[Cou19]	Correlated randomness	/	$O(\|C\|/\log\log\|C\|)$	/
This paper	Batch-Select	$O_\lambda(\|U_C\|)$	$\|\mathbf{x}_A\|+\|\mathbf{x}_B\|+O_\lambda(1)$	RLWE

Table 6: Efficiency comparison with existing 2PC protocols. We write $\mathbf{x}_A, \mathbf{x}_B, \mathbf{y}$ to mean the inputs to Alice and Bob, and the output. W.l.o.g. we assume $|\mathbf{x}_A| \le |\mathbf{x}_B|$. By U_C we denote a universal circuit (which can be made as small as $|C|\log|C|$ for arbitrary circuits C [ZYZL19]).

B Details on LEnc

We now prove correctness and security of Construction 1, LEnc.

Correctness. In the following analysis of LEnc, whenever the public parameters $\mathbf{b}_0, \mathbf{b}_1$ and the database $\mathbf{a} \in \mathcal{R}_q^w$ are fixed, we define (for any prefix $\mathsf{pre} \in \{0,1\}^{<\ell}$) the hash tree recursively as

$$y_{\mathsf{pre}} = \begin{pmatrix}\mathbf{b}_0^T & \mathbf{b}_1^T\end{pmatrix} \cdot \mathbf{z}_{\mathsf{pre}} \quad \text{where} \quad \mathbf{z}_{\mathsf{pre}} := \begin{pmatrix}-\mathbf{g}^{-1}(y_{\mathsf{pre}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{pre}\|1})\end{pmatrix} ,$$

and the leaves y_{ind} for $\mathsf{ind} \in \{0,1\}^\ell$ are given by $\mathbf{a}$.

We first show correctness with error bound $B_{\mathsf{LEnc}} = g \cdot m \cdot \gamma_{\mathcal{R}} \cdot \log_2 w \cdot s \cdot \sqrt{\lambda}$. Note that for any $\mathsf{ind} \in \{0,1\}^\ell$, evaluation yields

$$\begin{aligned}
\boldsymbol{\delta}[\mathsf{ind}] &= \sum_{i=0}^{\ell-1} \mathbf{C}_i[\mathsf{ind}] \cdot \ \mathbf{z}_{\mathsf{ind}_{:i}} \\
&= \sum_{i=0}^{\ell-1} \left(\mathbf{r}_i[\mathsf{ind}] \cdot \begin{pmatrix}\mathbf{b}_0^T & \mathbf{b}_1^T\end{pmatrix} + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \begin{pmatrix}\overline{\mathsf{ind}_i}\mathbf{g}^T & \mathsf{ind}_i\mathbf{g}^T\end{pmatrix} + \mathbf{E}_i[\mathsf{ind}] \right) \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \\
&= \sum_{i=0}^{\ell-1} \mathbf{r}[\mathsf{ind}] \cdot y_{\mathsf{ind}_{:i}} - \mathbf{r}_{i+1}[\mathsf{ind}] \cdot y_{\mathsf{ind}_{:i+1}} + \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \\
&= \mathbf{r}_0[\mathsf{ind}] \cdot y_\epsilon - \mathbf{s}[\mathsf{ind}] \cdot \mathbf{a}[\mathsf{ind}] + \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \ .
\end{aligned}$$

Therefore, we just need to prove that $\left\| \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \right\|_\infty < B_{\mathsf{LEnc}}$. Note that $\|\mathbf{E}_i\|_\infty \le \sqrt{\lambda} \cdot s$ and $\|\mathbf{z}_{\mathsf{ind}_{:i}}\|_\infty < g$. We get

$$\begin{aligned}
\left\| \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \right\|_\infty &\le \sum_{i=0}^{\ell-1} \sum_{j \in [m]} \|\mathbf{E}_i[\mathsf{ind}, j] \cdot \mathbf{z}_{\mathsf{ind}_{:i}}[j]\|_\infty \\
&\le \sum_{i=0}^{\ell-1} \sum_{j \in [m]} \gamma_{\mathcal{R}} \cdot \|\mathbf{E}_i\|_\infty \cdot \|\mathbf{z}_{\mathsf{ind}_{:i}}\|_\infty \\
&< \ell \cdot m \cdot \gamma_{\mathcal{R}} \cdot \sqrt{\lambda} \cdot s \cdot g = B_{\mathsf{LEnc}} \ .
\end{aligned}$$

Lemma 10 (Security of Construction 1). *Assuming* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$*, Construction 1 (*$\mathsf{LEnc}$*) fulfills* 2^λ*-simulation security with* T*-noise leakage (Definition 10).*

Proof. The simulator Sim takes the public parameters and the databases $(\mathsf{pp}, \{\mathbf{a}^{(t)}\})$, and simulates ciphertext and noise $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]})$ as follows:

- First, sample all ciphertexts purely random

$$\widetilde{\mathbf{C}}_i \leftarrow\!\$\ \mathcal{R}_q^{w \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1 \ ,$$

  and choose $\widetilde{\mathsf{ct}} := (\widetilde{\mathbf{C}}_0, \ldots, \widetilde{\mathbf{C}}_{\ell-1})$.
- Then, simulate the noise leakages as follows: sample

$$\mathbf{E}_i \leftarrow\!\$\ \chi^{w \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1 \ ,$$

and for each $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$, compute

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \ .$$

Return $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]})$.

In order to prove that the *real world* is indistinguishable from the *simulated world*, we define Hyb'_0 to output $((\mathbf{b}_0, \mathbf{b}_1), (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}), \{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}\}_{[T]})$, where these values are computed as follows:

$$\boxed{\mathbf{b}_0, \mathbf{b}_1} \leftarrow\!\$\ \mathcal{R}_q^m \tag{5}$$

For $i = 0, \ldots, \ell-1$ and $\mathsf{ind} \in \{0,1\}^\ell$:

$$\boxed{\mathbf{C}_i[\mathsf{ind}]} := \mathbf{r}_i[\mathsf{ind}] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \left(\overline{\mathsf{ind}_i} \cdot \mathbf{g}^T \ \mathsf{ind}_i \cdot \mathbf{g}^T\right) + \mathbf{E}_i[\mathsf{ind}] \tag{6}$$

$$\left| \begin{array}{l} \mathbf{r}_i[\mathsf{ind}] \leftarrow\!\$\ \mathcal{R}_q \text{ and } \mathbf{r}_\ell := \mathbf{s} \\ \mathbf{E}_i[\mathsf{ind}] \leftarrow\!\$\ \overline{\mathcal{D}}^{2m}_{\mathcal{R},s} \end{array} \right.$$

For $t \in [T]$ and $\mathsf{ind} \in \{0,1\}^\ell$:

$$\boxed{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}]} := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] \quad \left| \begin{array}{l} \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{0 \le i < \ell} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \\ \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] \leftarrow\!\$\ \overline{\chi}_{\mathsf{LEnc}} \end{array} \right. \tag{7}$$

Note that Hyb'_0 is identical to the real world. To see this, consider the real noise leakage $\mathbf{e}^{(t)} = \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)})$ with $\boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)})$. Then, as in the correctness analysis above which calculates $\boldsymbol{\delta}^{(t)}[\mathsf{ind}]$, we get

$$\mathbf{e}^{(t)}[\mathsf{ind}] = \boldsymbol{\delta}^{(t)}[\mathsf{ind}] - (\mathbf{r}[\mathsf{ind}] \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)}[\mathsf{ind}]) = \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} = \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}]$$

for any $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$.

As a first step, we define a hybrid Hyb_0 that is identical to the previous Hyb'_0, except that it samples the error matrices $\mathbf{E}_i[\mathsf{ind}]$ from the Gaussian $\chi^{2m} = \mathcal{D}^{2m}_{\mathcal{R},s}$ instead of the *truncated* Gaussian $\overline{\mathcal{D}}^{2m}_{\mathcal{R},s}$, and similarly $\overline{\mathbf{e}}^{(t)}[\mathsf{ind}]$ from the Gaussian $\mathcal{D}_{\mathcal{R},\overline{s}}$ instead of $\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$. By Lemma 2, we have $\mathsf{Hyb}_0 \approx^{2^\lambda}_{\mathrm{s}} \mathsf{Hyb}'_0$.

Next, for any $i^* \in [\ell]$, we define Hyb_{i^*} to be identical to Hyb_0, except that the ciphertexts of the first i^* layers are sampled as

$$\widetilde{\mathbf{C}}_i \leftarrow\!\$\ \mathcal{R}_q^{w \times 2m} \quad \text{for } i < i^* \ .$$

Note that Hyb_ℓ is identical to the simulated world.

It remains to show that $\mathsf{Hyb}_{i^*} \approx^{2^\lambda}_{\mathrm{c}} \mathsf{Hyb}_{i^*+1}$ for any $i^* = 0, \ldots, \ell-1$. To do so, we further divide the hybrid Hyb_{i^*} into several sub-hybrids $\mathsf{Hyb}_{i^*,j}$ for $0 \le j \le w$, which are identical to Hyb_{i^*}, except that $\mathbf{C}_{i^*}[\mathsf{ind}]$ is sampled as

$$\widetilde{\mathbf{C}}_{i^*}[\mathsf{ind}] \leftarrow\!\$\ \mathcal{R}_q^{2m} \quad \text{for } j' < j \ ,$$

where ind is the bitstring representing j'. We can see that $\mathsf{Hyb}_{i^*} \equiv \mathsf{Hyb}_{i^*,0}$ and that $\mathsf{Hyb}_{i^*,w} \equiv \mathsf{Hyb}_{i^*+1}$, and therefore it just remains to show that $\mathsf{Hyb}_{i^*,j} \approx_c^{2^\lambda} \mathsf{Hyb}_{i^*,j+1}$ for any $0 \le j < w$.

We do so by a reduction to the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$ assumption. Specifically, assuming an adversary $\mathcal{D}$ that distinguishes between $\mathsf{Hyb}_{i^*,j}$ and $\mathsf{Hyb}_{i^*,j+1}$, we construct an adversary $\mathcal{A}$ for elLWE as follows (where the running time of $\mathcal{A}$ additively increases by $\mathrm{poly}(\lambda)$ when compared to that of $\mathcal{D}$). We use $\mathsf{ind} \in \{0,1\}^\ell$ as the bitstring representing the integer j.

- After receiving public parameters $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$, choose the *leakage matrix* $\mathbf{Z} \in \mathcal{R}_q^{T \times 2m}$ in such a way that the t-th row is equal to the transpose of $\mathbf{z}^{(t)}_{\mathsf{ind}_{:i^*}}$.
- After receiving the LWE sample $\mathbf{y} \in \mathcal{R}_q^{2m}$ and leakage $\mathbf{l} \in \mathcal{R}_q^T$, run and output the result of $\mathcal{D}$ on a simulation of $\mathsf{Hyb}_{i^*,j}$, with two modifications:
  - The ind-th row of $\mathbf{C}_{i^*}$ is replaced by
$$\mathbf{C}_{i^*}[\mathsf{ind}] := \mathbf{y}^T + \mathbf{r}_{i^*+1}[\mathsf{ind}] \cdot \left(\overline{\mathsf{ind}_{i^*}} \cdot \mathbf{g}^T \; \mathsf{ind}_{i^*} \cdot \mathbf{g}^T\right)$$
    instead of being computed from the key $\mathbf{r}_{i^*}[\mathsf{ind}]$.
  - For every $t \in [T]$, the ind-th error is replaced by
$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{\substack{0 \le i < \ell \\ i \ne i^*}} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \quad \text{and} \quad \widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}] := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \mathbf{l}[t] \; .$$

The error-leakage Ring-LWE experiment with adversary $\mathcal{A}$ has the following distributions:

- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$. Here, $\mathbf{y}$ corresponds to a real LWE sample, i.e., $\mathbf{y}^T = \mathbf{r}_{i^*}[\mathsf{ind}] \cdot \left(\mathbf{b}_0^T \; \mathbf{b}_1^T\right) + \mathbf{E}_{i^*}[\mathsf{ind}]$ with leakage $\mathbf{l}[t] = \mathbf{E}_{i^*}[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i^*}} + \overline{e}^{(t)}[\mathsf{ind}]$ (for some fresh key $\mathbf{r}_{i^*}[\mathsf{ind}] \leftarrow\!\$\; \mathcal{R}_q$ and errors $\mathbf{E}_{i^*}[\mathsf{ind}] \leftarrow\!\$\; \chi^{2m}$ and $\overline{e}^{(t)}[\mathsf{ind}] \leftarrow\!\$\; \mathcal{D}_{\mathcal{R},\bar{s}}$).
  Therefore, $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j}$.
- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$. Here, $\mathbf{y} \leftarrow\!\$\; \mathcal{R}_q^{2m}$ is uniformly random and the leakage is equal to $\mathbf{l}[t] = \mathbf{E}_{i^*}[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} + \overline{e}^{(t)}[\mathsf{ind}]$ (for fresh errors $\mathbf{E}_{i^*}[\mathsf{ind}] \leftarrow\!\$\; \chi^{2m}$ and $\overline{e}^{(t)}[\mathsf{ind}] \leftarrow\!\$\; \mathcal{D}_{\mathcal{R},\bar{s}}$).
  Note that $\mathbf{C}_{i^*}[\mathsf{ind}]$ will be uniformly random (due to the randomness of $\mathbf{y}$). Therefore, $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j+1}$.

Therefore, the distinguisher $\mathcal{D}$ has exactly the same advantage as the adversary $\mathcal{A}$. By the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$ assumption, this implies that $\mathsf{Hyb}_{i^*,j} \approx_c^{2^\lambda} \mathsf{Hyb}_{i^*,j+1}$. Our hybrid argument thus shows $\mathsf{Hyb}'_0 \approx_c^{2^\lambda} \mathsf{Hyb}_\ell$, which concludes the security proof. □

B.1 Construction of Weak Linear Laconic Encryption

We now present a slight modification of Construction 1, to give a *weak* LEnc over a ring $\mathcal{R}_q$ (see Definition 11). It utilizes the same parameters as Construction 1 and has the same correctness error. However, simulation security with T-noise leakage w.r.t. noise distribution $\overline{\chi}_{\mathsf{LEnc}}$ requires the assumption of $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{Tw,2m}(g)}$.

Construction The main motivation is the observation that in Construction 1, for *every* single $\mathsf{ind} \in \{0,1\}^\ell$, we created *independent* ciphertexts allowing us to obtain $\mathbf{r}[\mathsf{ind}] \cdot y_\epsilon - \mathbf{s}[\mathsf{ind}] \cdot \mathbf{a}[\mathsf{ind}]$.

Hence, a natural question to ask is whether there is potential for optimization by exploiting the repeated usage of the same functionality.

Indeed, we are able to construct an optimized *weak* LEnc, i.e., when the output keys $\mathbf{s}$ provided to the encryption algorithm $\mathsf{LEnc.Enc}(\mathbf{s})$ are guaranteed to consist of w *identical* ring elements ($\mathbf{s} = s \cdot \mathbf{1}_w$ for some $s \in \mathcal{R}_q$). The size of the ciphertext will be reduced by a factor of $\frac{\ell}{2} = \frac{\log w}{2}$. However, it also requires a slightly stronger assumption of elLWE in which the leakage matrix has dimensions $Tw \times 2m$ instead of $T \times 2m$.

To get some intuition of where the improvement is coming from, consider the ciphertext $\mathsf{ct} = (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$ in Construction 1. Taking a closer look at $\mathbf{C}_{\ell-1}$, we can see that it will be equal to

$$\mathbf{C}_{\ell-1} = \begin{pmatrix} \mathbf{r}_{\ell-1}[1] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \mathbf{s}[1] \cdot \left(\mathbf{g}^T \ \mathbf{0}^T\right) \\ \mathbf{r}_{\ell-1}[2] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \mathbf{s}[2] \cdot \left(\mathbf{0}^T \ \mathbf{g}^T\right) \\ \vdots \\ \mathbf{r}_{\ell-1}[w-1] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \mathbf{s}[w-1] \cdot \left(\mathbf{g}^T \ \mathbf{0}^T\right) \\ \mathbf{r}_{\ell-1}[w] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \mathbf{s}[w] \cdot \left(\mathbf{0}^T \ \mathbf{g}^T\right) \end{pmatrix} + \mathbf{E}_{\ell-1} \ .$$

Now, because $\mathbf{s}$ only consists of identical elements s, the "ciphertexts" corresponding to rows $1, 3, 5, \ldots$ are all used to hide the same vector $s \cdot \left(\mathbf{g}^T \ \mathbf{0}^T\right)$. Additionally, the ciphertexts corresponding to rows $2, 4, 6, \ldots$ are all used to hide the same vector $s \cdot \left(\mathbf{0}^T \ \mathbf{1}^T\right)$.

To exploit this pattern, we will sample $\mathbf{r}_{\ell-1}$ in such a way that $\mathbf{r}_{\ell-1}[1] = \mathbf{r}_{\ell-1}[3] = \ldots$ and $\mathbf{r}_{\ell-1}[2] = \mathbf{r}_{\ell-1}[4] = \ldots$, which allows us to use a "compressed" $\mathbf{C}_{\ell-1}$ of the form

$$\widehat{\mathbf{C}}_{\ell-1} = \begin{pmatrix} \widehat{\mathbf{r}}_{\ell-1}[1] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + s \cdot \left(\mathbf{g}^T \ \mathbf{0}^T\right) \\ \widehat{\mathbf{r}}_{\ell-1}[2] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + s \cdot \left(\mathbf{0}^T \ \mathbf{g}^T\right) \end{pmatrix} + \widehat{\mathbf{E}}_{\ell-1} \ ,$$

which consists only of two rows. We get the "decompressed" version by computing

$$\mathbf{C}_{\ell-1} = \mathbf{1}_{2^{\ell-1}} \otimes \widehat{\mathbf{C}}_{\ell-1} \ ,$$

and they correspond to keys $\mathbf{r}_{\ell-1} = \mathbf{1}_{2^{\ell-1}} \otimes \widehat{\mathbf{r}}_{\ell-1}$.

Note that we reduced the number of rows in $\widehat{\mathbf{C}}_{\ell-1}$ to just 2. This, in turn, can be used to see that in $\mathbf{C}_{\ell-2}$, only 4 different patterns need to be encrypted, and so on. In general, through this compression trick and reusing the same keys $\widehat{\mathbf{r}}_i \in \mathcal{R}_q^{\frac{w}{2^i} \times 2m}$ for 2^i times, $\widehat{\mathbf{C}}_i$ will consist of $\frac{w}{2^i}$ rows. The only potential issue with this approach is that the same noise, e.g. $\widehat{\mathbf{E}}_{\ell-1}$, is leaked w times, and therefore we will slightly modify the security proof and depend on a variant of elLWE that allows more leakage than before.

The resulting construction differs from that in Section B only in the encryption and evaluation algorithms:

Construction 5 (Weak Linear Laconic Encryption).

$\underline{\mathsf{Enc}(\mathbf{s} = s \cdot \mathbf{1}_w) \to \mathbf{r}, \mathsf{ct}}$**:** Sample Ring-LWE secrets $\widehat{\mathbf{r}}_i \leftarrow\!\$\ \mathcal{R}_q^{\frac{w}{2^i}}$ for each layer $i = 0, \ldots, \ell - 1$. For notational convenience, set $\widehat{\mathbf{r}}_\ell := s$. Also sample *truncated* Ring-LWE noises $\widehat{\mathbf{E}}_i \leftarrow\!\$\ \chi^{\frac{w}{2^i} \times 2m}$. Then, for each of the $\frac{w}{2^i}$ rows, indexed by $\mathsf{suf} \in \{0,1\}^{\ell-i}$, compute for each $i = 0, \ldots, \ell - 1$ the compressed ciphertext

$$\widehat{\mathbf{C}}_i[\mathsf{suf}] := \widehat{\mathbf{r}}_i[\mathsf{suf}] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \widehat{\mathbf{r}}_{i+1}[\mathsf{suf}_{1:}] \cdot \left(\overline{\mathsf{suf}_0} \cdot \mathbf{g}^T \ \mathsf{suf}_0 \cdot \mathbf{g}^T\right) + \widehat{\mathbf{E}}_i[\mathsf{suf}] \ .$$

We write the results as ℓ matrices $\widehat{\mathbf{C}}_i \in \mathcal{R}_q^{\frac{w}{2^i} \times 2m}$.

Return input keys $\mathbf{r} := \mathbf{r}_0$ and ciphertexts $\mathsf{ct} := (\widehat{\mathbf{C}}_0, \ldots, \widehat{\mathbf{C}}_{\ell-1})$.

$\underline{\mathsf{Eval}(\mathsf{ct}, \mathbf{a}) \to \boldsymbol{\delta}}$**:** Parse $\mathsf{ct} := (\widehat{\mathbf{C}}_0, \ldots, \widehat{\mathbf{C}}_{\ell-1})$. Decompress these ciphertexts in the following way:

$$\mathbf{C}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{C}}_i \in \mathcal{R}_q^{w \times 2m} .$$

As in Digest, compute y_{pre} for all $\mathsf{pre} \in \{0,1\}^{\leq \ell}$ from database $\mathbf{a}$.
Then, for each $\mathsf{ind} \in \{0,1\}^\ell$, compute the result

$$\boldsymbol{\delta}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{C}_i[\mathsf{ind}] \cdot \begin{pmatrix} -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|1}) \end{pmatrix} ,$$

and return the vector $\boldsymbol{\delta}$.

Correctness. Correctness of Construction 5 follows from the fact that the decompressed ciphertexts are equal to "normal" ciphertexts as created in Construction 1:

$$\begin{aligned} \mathbf{C}_i[\mathsf{ind}] &= \widehat{\mathbf{C}}_i[\mathsf{ind}_{i:}] \\ &= \widehat{\mathbf{r}}_i[\mathsf{ind}_{i:}] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \widehat{\mathbf{r}}_{i+1}[\mathsf{ind}_{i+1:}] \cdot \left(\overline{\mathsf{ind}_i} \cdot \mathbf{g}^T \ \mathsf{ind}_i \cdot \mathbf{g}^T\right) + \widehat{\mathbf{E}}_i[\mathsf{ind}_{i:}] \\ &= \mathbf{r}_i[\mathsf{ind}] \cdot \left(\mathbf{b}_0^T \ \mathbf{b}_1^T\right) + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \left(\overline{\mathsf{ind}_i} \cdot \mathbf{g}^T \ \mathsf{ind}_i \cdot \mathbf{g}^T\right) + \mathbf{E}_i[\mathsf{ind}] , \end{aligned}$$

where

$$\mathbf{r}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{r}}_i \quad \text{and} \quad \mathbf{E}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{E}}_i$$

denote the "decompressed" keys and noise. Now we can apply correctness of the original Construction 1 (which does make any assumptions on entries of $\mathbf{r}_i$ or $\mathbf{E}_i$ being independent of each other).

Lemma 11 (Security of Construction 5). *Assuming* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{Tw,2m}(g)}$*, Construction 1 fulfills* 2^λ-simulation security with T-noise leakage *(Definition 10).*

Proof. The simulator Sim takes the public parameters and the databases $(\mathsf{pp}, \{\mathbf{a}^{(t)}\})$, and simulates ciphertext and noise $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[}T])$ as follows:

- First, sample all compressed ciphertexts purely random

$$\widetilde{\widehat{\mathbf{C}}}_i \leftarrow\!\$\ \mathcal{R}_q^{\frac{w}{2^i} \times 2m} \quad \text{for } i = 0, \ldots, \ell-1 ,$$

  and choose $\widetilde{\mathsf{ct}} := (\widetilde{\widehat{\mathbf{C}}}_0, \ldots, \widetilde{\widehat{\mathbf{C}}}_{\ell-1})$.
- Then, simulate the noise leakages as follows: sample

$$\widehat{\mathbf{E}}_i \leftarrow\!\$\ \chi^{\frac{w}{2^i} \times 2m} \quad \text{for } i = 0, \ldots, \ell-1 ,$$

  "decompress" these noise matrices by choosing

$$\mathbf{E}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{E}}_i \quad \text{for } i = 0, \ldots, \ell-1 ,$$

  and for each $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$, compute

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} .$$

  Return $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]})$.

In order to prove that the *real world* is indistinguishable from the *simulated world*, we define Hyb'_0 to output $((\mathbf{b}_0, \mathbf{b}_1), (\widehat{\mathbf{C}}_0, \ldots, \widehat{\mathbf{C}}_{\ell-1}), \{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}\}_{[T]})$, where these values are computed as follows:

$$\boxed{\mathbf{b}_0, \mathbf{b}_1} \leftarrow\!\$\ \mathcal{R}_q^m$$

For $i = 0, \ldots, \ell - 1$ and $\mathsf{suf} \in \{0,1\}^{\ell - i}$:

$$\boxed{\widehat{\mathbf{C}}_i[\mathsf{suf}]} := \widehat{\mathbf{r}}_i[\mathsf{suf}] \cdot \left(\mathbf{b}_0^T\ \mathbf{b}_1^T\right) + \widehat{\mathbf{r}}_{i+1}[\mathsf{suf}_{1:}] \cdot \left(\overline{\mathsf{suf}_0} \cdot \mathbf{g}^T\ \mathsf{suf}_0 \cdot \mathbf{g}^T\right) + \widehat{\mathbf{E}}_i[\mathsf{suf}]$$

$$\left|\begin{array}{l} \mathbf{r}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{r}}_i \quad \text{for } i = i^*, \ldots, \ell \text{ with } \widehat{\mathbf{r}}_i \leftarrow\!\$\ \mathcal{R}_q^{\frac{w}{2^i}} \text{ and } \widehat{\mathbf{r}}_\ell := s \\ \mathbf{E}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{E}}_i \quad \text{for } i = 0, \ldots, \ell - 1 \text{ with } \widehat{\mathbf{E}}_i \leftarrow\!\$\ \overline{\mathcal{D}}_{\mathcal{R},s}^{\frac{w}{2^i} \times 2m} \end{array}\right.$$

For $t \in [T]$ and $\mathsf{ind} \in \{0,1\}^\ell$:

$$\boxed{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}]} := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] \quad \left|\begin{array}{l} \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{0 \le i < \ell} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \\ \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] \leftarrow\!\$\ \overline{\chi}_{\mathsf{LEnc}} \end{array}\right.$$

Note that Hyb'_0 is identical to the real world. To see this, consider the real noise leakage $\mathbf{e}^{(t)} = \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)})$ with $\boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)})$. Then, as in the correctness analysis above which calculates $\boldsymbol{\delta}^{(t)}[\mathsf{ind}]$, we get

$$\mathbf{e}^{(t)}[\mathsf{ind}] = \boldsymbol{\delta}^{(t)}[\mathsf{ind}] - (\mathbf{r}[\mathsf{ind}] \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)}[\mathsf{ind}]) = \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} = \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}]$$

for any $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$.

As a first step, we define a hybrid Hyb_0 that is identical to the previous Hyb'_0, except that it samples the error matrices $\widehat{\mathbf{E}}_i[\mathsf{suf}]$ from the Gaussian $\chi^{2m} = \mathcal{D}^{2m}_{\mathcal{R},s}$ instead of the *truncated* Gaussian $\overline{\mathcal{D}}^{2m}_{\mathcal{R},s}$, and similarly $\overline{\mathbf{e}}^{(t)}[\mathsf{ind}]$ from the Gaussian $\mathcal{D}_{\mathcal{R},\bar{s}}$ instead of $\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\bar{s}}$. By Lemma 2, we have $\mathsf{Hyb}_0 \approx_{\mathrm{s}}^{2^\lambda} \mathsf{Hyb}'_0$.

Next, for any $i^* \in [\ell]$, we define Hyb_{i^*} to be identical to Hyb_0, except that the ciphertexts of the first i^* layers are sampled as

$$\widetilde{\widehat{\mathbf{C}}}_i \leftarrow\!\$\ \mathcal{R}_q^{\frac{w}{2^i} \times 2m} \quad \text{for } i < i^* \ .$$

Note that Hyb_ℓ is identical to the simulated world.

It remains to show that $\mathsf{Hyb}_{i^*} \approx_{\mathrm{c}} \mathsf{Hyb}_{i^*+1}$ for any $i^* = 0, \ldots, \ell - 1$. To do so, we further divide the hybrid Hyb_{i^*} into several sub-hybrids $\mathsf{Hyb}_{i^*,j}$ for $0 \le j \le \frac{w}{2^{i^*}}$, which are identical to Hyb_{i^*}, except that $\widehat{\mathbf{C}}_{i^*}[\mathsf{ind}]$ is sampled as

$$\widetilde{\widehat{\mathbf{C}}}_{i^*}[\mathsf{suf}] \leftarrow\!\$\ \mathcal{R}_q^{2m} \quad \text{for } j' < j \ ,$$

where suf is the bitstring representing j'. We can see that $\mathsf{Hyb}_{i^*} \equiv \mathsf{Hyb}_{i^*,0}$ and that $\mathsf{Hyb}_{i^*,w} \equiv \mathsf{Hyb}_{i^*+1}$, and therefore it just remains to show that $\mathsf{Hyb}_{i^*,j} \approx_{\mathrm{c}} \mathsf{Hyb}_{i^*,j+1}$ for any $0 \le j < w$.

We do so by a reduction to the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T2^{i^*},2m}}(g)$ assumption. Specifically, assuming an adversary $\mathcal{D}$ that distinguishes between $\mathsf{Hyb}_{i^*,j}$ and $\mathsf{Hyb}_{i^*,j+1}$, we construct an adversary $\mathcal{A}$ for elLWE as follows (where the running time of $\mathcal{A}$ additively increases by $\mathrm{poly}(\lambda)$ when compared to that of $\mathcal{D}$). We use $\mathsf{suf} \in \{0,1\}^{\ell - i}$ as the bitstring representing the integer j.

– After receiving public parameters $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$, choose the *leakage matrix* $\mathbf{Z} \in \mathcal{R}_q^{T2^{i^*} \times 2m}$, with rows indexed by (t, pre), for $t \in [T]$ and $\mathsf{pre} \in \{0,1\}^{i^*}$, in such a way that the (t, pre)-th row is equal to the transpose of $\mathbf{z}_{\mathsf{pre}}^{(t)}$.
– After receiving the LWE sample $\mathbf{y} \in \mathcal{R}_q^{2m}$ and leakage $\mathbf{l} \in \mathcal{R}_q^{T2^{i^*}}$, run and output the result of $\mathcal{D}$ on a simulation of $\mathsf{Hyb}_{i^*,j}$, with two modifications:
  - The suf-th row of $\widehat{\mathbf{C}}_{i^*}$ is replaced by

$$\widehat{\mathbf{C}}_{i^*}[\mathsf{suf}] := \mathbf{y}^T + \widehat{\mathbf{r}}_{i^*+1}[\mathsf{suf}_{1:}] \cdot \left(\overline{\mathsf{suf}_0} \cdot \mathbf{g}^T \; \mathsf{suf}_0 \cdot \mathbf{g}^T\right)$$

    instead of being computed from the key $\widehat{\mathbf{r}}_{i^*}[\mathsf{suf}]$.
  - For every $t \in [T]$, and every $\mathsf{pre} \in \{0,1\}^{i^*}$, the $\mathsf{ind} = \mathsf{pre}\|\mathsf{suf}$-th error is replaced by

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{\substack{0 \le i < \ell \\ i \neq i^*}} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind},i}^{(t)} \quad \text{and} \quad \widetilde{\mathbf{e}}_{\mathsf{res}}^{(t)}[\mathsf{ind}] := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \mathbf{l}[t, \mathsf{pre}] \; .$$

The error-leakage Ring-LWE experiment with adversary $\mathcal{A}$ has the following distributions:

– Consider the experiment $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$. Here, $\mathbf{y}$ corresponds to a real LWE sample, i.e., $\mathbf{y}^T = \widehat{\mathbf{r}}_{i^*}[\mathsf{suf}] \cdot \left(\mathbf{b}_0^T \; \mathbf{b}_1^T\right) + \widehat{\mathbf{E}}_{i^*}[\mathsf{suf}]$ with leakage $\mathbf{l}[t, \mathsf{pre}] = \widehat{\mathbf{E}}_{i^*}[\mathsf{suf}] \cdot \mathbf{z}_{\mathsf{pre}}^{(t)} + \overline{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}]$ (for some fresh key $\widehat{\mathbf{r}}_{i^*}[\mathsf{suf}] \leftarrow\!\$\; \mathcal{R}_q$ and errors $\widehat{\mathbf{E}}_{i^*}[\mathsf{suf}] \leftarrow\!\$\; \chi^{2m}$ and $\overline{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}] \leftarrow\!\$\; \mathcal{D}_{\mathcal{R},\overline{s}}$).
Therefore, $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j}$.
– Consider the experiment $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$. Here, $\mathbf{y} \leftarrow\!\$\; \mathcal{R}_q^{2m}$ is uniformly random and the leakage is equal to $\mathbf{l}[t, \mathsf{pre}] = \widehat{\mathbf{E}}_{i^*}[\mathsf{suf}] \cdot \mathbf{z}_{\mathsf{pre}}^{(t)} + \overline{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}]$ (for fresh errors $\mathbf{E}_{i^*}[\mathsf{suf}] \leftarrow\!\$\; \chi^{2m}$ and $\overline{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}] \leftarrow\!\$\; \mathcal{D}_{\mathcal{R},\overline{s}}$).
Note that $\mathbf{C}_{i^*}[\mathsf{ind}]$ will be uniformly random (due to the randomness of $\mathbf{y}$). Therefore, $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j+1}$.

Therefore, the distinguisher $\mathcal{D}$ has exactly the same advantage as the adversary $\mathcal{A}$. By the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{Tw,2m}(g)}$ assumption, this implies that $\mathsf{Hyb}_{i^*,j} \approx_{\mathrm{c}}^{2^\lambda} \mathsf{Hyb}_{i^*,j+1}$. Our hybrid argument thus shows $\mathsf{Hyb}'_0 \approx_{\mathrm{c}}^{2^\lambda} \mathsf{Hyb}_\ell$, which concludes the security proof. □

B.2 LEnc Parameter Setting

We now prove the claimed efficiencies for the parameter setting described in Lemma 5.

Proof (of Lemma 5). Under assumption $\mathsf{LWE}_{\mathcal{R},O(1),q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$, we may conclude (using Theorem 2 and Lemma 3) that also the error-leakage version $\mathsf{elLWE}_{\mathcal{R},q,\mathcal{D}_{\mathcal{R},s},\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$ holds, with parameters $s = 3q^{0.1}$ and $\overline{s} = 2s \cdot g \cdot n\sqrt{2Tm}$.

To obtain the claimed parameters, we choose the gadget-base $g = \lceil q^{0.05} \rceil$ (as used inside of our LEnc construction) and hence $m \le 20$. Thus, we get

$$\begin{aligned} \max \overline{\chi}_{\mathsf{LEnc}} = \sqrt{\lambda} \cdot \overline{s} &= q^{0.1} \cdot \underbrace{g}_{\le 2q^{0.05}} \cdot \underbrace{n}_{\le q^{1/15}} \cdot \underbrace{\sqrt{T}}_{\le q^{1/4}} \cdot \underbrace{\sqrt{m}}_{\le 20} \cdot \underbrace{\sqrt{\lambda}}_{\le q^{1/30}} \cdot 6\sqrt{2} \\ &\le 1000\sqrt{q} \; , \end{aligned}$$

and similarly

$$B_{\mathsf{LEnc}} = \underbrace{g}_{2q^{0.05}} \cdot \underbrace{m}_{\leq 20} \cdot \underbrace{\gamma_{\mathcal{R}}}_{\leq q^{1/15}} \cdot \underbrace{\log_2 w}_{\leq \lambda \leq q^{1/15}} \cdot q^{0.1} \cdot \underbrace{\sqrt{\lambda}}_{\leq q^{1/30}} \cdot 2$$
$$\leq 80q^{0.35} \leq \sqrt{q} \; .$$

Similarly, we also get a *weak* LEnc scheme with $T = \frac{\sqrt{q}}{w}$-noise leakage under the same parameters, because the $\mathsf{elLWE}_{\mathcal{R},q,\mathcal{D}_{\mathcal{R},s},\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{Tw,2m}(g)}$ assumption holds under $\mathsf{LWE}_{\mathcal{R},O(1),q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$, with the same parameters as above: $s = 3q^{0.1}$ and $\overline{s} = 2s \cdot g \cdot n\sqrt{2m} \cdot q^{1/4}$. □

C Details on LHE

We now prove correctness and security of Construction 2, LHE.

Correctness. We verify the decryption process, and show that the magnitude of the noise is bounded by $B_{\mathsf{LHE}}(\lambda) = (s \cdot m \cdot \gamma_{\mathcal{R}} \cdot g + \overline{s}) \cdot \sqrt{\lambda}$:

$$\begin{aligned}
\mathbf{m}_{\mathsf{res}} &= \overbrace{\mathsf{ct}_1 \cdot \mathbf{g}^{-1}(y) + \mathsf{ct}_2}^{\mathsf{ct}_{\mathsf{res}}} - \mathbf{a} \cdot \overbrace{(\mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2)}^{\mathsf{sk}_y} \\
&= (\mathbf{m}_1 \cdot y + \mathbf{m}_2) + (\mathbf{E} \cdot \mathbf{g}^{-1}(y) + \overline{\mathbf{e}}) \; , \\
\left\|\mathbf{E} \cdot \mathbf{g}^{-1}(y) + \overline{\mathbf{e}}\right\|_\infty &\leq m \cdot \gamma_{\mathcal{R}} \cdot \|\mathbf{E}\|_\infty \cdot \left\|\mathbf{g}^{-1}(y)\right\|_\infty + \|\overline{\mathbf{e}}\|_\infty \\
&< m \cdot \gamma_{\mathcal{R}} \cdot g \cdot \max \overline{D}_{\mathcal{R},s} + \max \overline{D}_{\mathcal{R},\overline{s}} \\
&< (m \cdot \gamma_{\mathcal{R}} \cdot g \cdot s + \overline{s}) \cdot \sqrt{\lambda} = B_{\mathsf{LHE}} \; .
\end{aligned}$$

Lemma 12 (Security of Construction 2). *Assuming* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$, *Construction 2 (*$\mathsf{LHE}$*) fulfills* T-times 2^λ-simulation security *(Definition 13).*

Proof. The simulator Sim takes the public parameters pp, the evaluated messages and ring elements $\{\mathbf{m}^{(t)}_{\mathsf{res}}, y^{(t)}\}$, and simulates ciphertexts and decryption keys $(\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}, \mathsf{sk}_y^{(t)}\})$ as follows:

– Sample the first ciphertext ct_1 and all decryption keys $\mathsf{sk}_y^{(t)}$ at random:

$$\widetilde{\mathsf{ct}}_1 \leftarrow\!\$\; \mathcal{R}_q^{w \times m} \; , \quad \widetilde{\mathsf{sk}}_y^{(t)} \leftarrow\!\$\; \mathcal{R}_q \quad \forall t \in [T] \; .$$

– Then, simulate the remaining ciphertexts $\mathsf{ct}_2^{(t)}$ by sampling noises $\mathbf{E} \leftarrow \chi^{w \times m}$ and $\overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}^w$ as in an honest execution (except that they are not truncated), and computing

$$\begin{aligned}
\widetilde{\mathsf{ct}}^{(t)}_{\mathsf{res}} &:= \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}^{(t)}_{\mathsf{res}} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \quad \forall t \in [T] \\
\widetilde{\mathsf{ct}}^{(t)}_2 &:= \widetilde{\mathsf{ct}}^{(t)}_{\mathsf{res}} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)}) \quad \forall t \in [T]
\end{aligned}$$

We show a series of hybrid experiments that transitions from Hyb_0 (the "real" distribution as defined in Definition 13) to Hyb_4 (the "simulated" distribution). We abuse notation to also write Hyb_i as the output of the distribution of the corresponding experiment.

Hyb_0 To recall, Hyb_0 generates ciphertexts $\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}\}$ and decryption keys $\{\mathsf{sk}_y^{(t)}\}$ in the following way:

$$\boxed{\mathbf{a}} \leftarrow\$ \mathcal{R}_q^m \tag{8}$$

$$\boxed{\mathsf{ct}_1} := \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \Big| \quad \begin{array}{l} \mathbf{s}_1 \leftarrow\$ \mathcal{R}_q^m \\ \mathbf{E} \leftarrow\$ \overline{\mathcal{D}}_{\mathcal{R},s}^{w \times m} \end{array} \tag{9}$$

For $t \in [T]$:

$$\boxed{\mathsf{ct}_2^{(t)}} := \mathbf{a} \cdot s_2^{(t)} + \mathbf{m}_2^{(t)} + \overline{\mathbf{e}}^{(t)} \quad \Big| \quad \begin{array}{l} s_2^{(t)} \leftarrow\$ \mathcal{R}_q \\ \overline{\mathbf{e}}^{(t)} \leftarrow\$ \overline{\mathcal{D}}_{\mathcal{R},\bar{s}}^{w} \end{array} \tag{10}$$

$$\boxed{\mathsf{sk}_y^{(t)}} := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)} \tag{11}$$

Hyb_1 As a first step, in Hyb_1 we sample the errors $\mathbf{E}$ and $\overline{\mathbf{e}}^{(t)}$ as Gaussians from $\chi^{w \times m} = \mathcal{D}_{\mathcal{R},s}^{w \times m}$ and $\overline{\chi}^w = \mathcal{D}_{\mathcal{R},\bar{s}}^{w}$, instead of truncated Gaussians from $\overline{\mathcal{D}}_{\mathcal{R},s}^{w \times m}$ and $\overline{\mathcal{D}}_{\mathcal{R},\bar{s}}^{w}$, respectively. By Lemma 2, we have $\mathsf{Hyb}_1 \approx_{\mathrm{s}}^{2^\lambda} \mathsf{Hyb}_0$.

Hyb_2 Now, we switch the order of computation: instead of computing $\mathsf{ct}_2^{(t)}$ directly from $\mathbf{m}_2^{(t)}$, we first define the *combined* ciphertext $\mathsf{ct}_{\mathsf{res}}^{(t)}$ (which is an encryption of $\mathbf{m}_{\mathsf{res}}$ under $\mathsf{sk}_y^{(t)}$), and then simulate $\mathsf{ct}_2^{(t)}$ as a combination of $\mathsf{ct}_{\mathsf{res}}^{(t)}$ and ct_1.

More precisely, we leave ct_1 and $\mathsf{sk}_y^{(t)}$ unchanged from the previous hybrid, but instead compute $\widetilde{\mathsf{ct}}_2^{(t)}$ as follows:

$$\boxed{\mathsf{sk}_y^{(t)}} := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)} \quad \Big| \quad s_2^{(t)} \leftarrow\$ \mathcal{R}_q$$

$$\boxed{\widetilde{\mathsf{ct}}_2^{(t)}} := \widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)}) \quad \Big| \quad \begin{array}{l} \overline{\mathbf{e}}^{(t)} \leftarrow\$ \overline{\chi}^w \\ \widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \end{array}$$

By definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$ and $\mathbf{m}_{\mathsf{res}}^{(t)} = \mathbf{m}_1 \cdot y^{(t)} + \mathbf{m}_2^{(t)}$, this way of defining ct_2 is identical to the previous one. Therefore, we have $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$.

Hyb_3 Note that in the previous hybrid, the uniformly random key $s_2^{(t)}$ is used nowhere but in the definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$. Therefore, we may equivalently generate

$$\boxed{\widetilde{\mathsf{sk}}_y^{(t)}} \leftarrow\$ \mathcal{R}_q \quad \forall t \in [T]$$

uniformly random instead of computing it from $s_2^{(t)}$. We get $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 In this hybrid, we replace the first ciphertext $\mathsf{ct}_1 = \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E}$ by a uniformly generated vector

$$\widetilde{\mathsf{ct}}_1 \leftarrow\$ \mathcal{R}_q^{w \times m} .$$

While we would like to use the Ring-LWE assumption for this step, note that the noise matrix $\mathbf{E}$ is still used in the definition of $\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)}$. Therefore, we need to use the more powerful elLWE, which allows us to utilize the noise leakage $\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}$.

In more detail, we define sub-hybrids $\mathsf{Hyb}_{3,i}$ for $i = 0, \ldots, m$, where $\mathsf{Hyb}_{3,i}$ is identical to Hyb_3, except that the first i columns of ct_1 are sampled uniformly random. Note that $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_{3,0}$

and $\mathsf{Hyb}_{3,m} \equiv \mathsf{Hyb}_4$. Therefore, in order to prove $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$, it only remains to show $\mathsf{Hyb}_{3,i-1} \approx_c \mathsf{Hyb}_{3,i}$ for $i = 1, \ldots, m$. To do so, assume there exists a distinguisher $\mathcal{D}$ for $\mathsf{Hyb}_{3,i-1}$ and $\mathsf{Hyb}_{3,i}$. We construct an adversary $\mathcal{A}$ for $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$ in the following way (where the running time of $\mathcal{A}$ additively increases by $\mathrm{poly}(\lambda)$ when compared to that of $\mathcal{D}$):

- After receiving public parameters $\mathbf{a} \in \mathcal{R}_q^w$, choose the *leakage matrix* $\mathbf{Z} \in \mathcal{R}_q^{Tw \times w}$ in the following way. Let $\mathbf{g}^{-1}(y^{(t)})[i]$ be the i-th entry of the decomposition $\mathbf{g}^{-1}(y^{(t)})$. Then, select
$$\mathbf{z}^T := \left(\mathbf{g}^{-1}(y^{(1)})[i] \ldots \mathbf{g}^{-1}(y^{(T)})[i]\right)$$
and choose
$$\mathbf{Z} := \mathrm{diag}(\underbrace{\mathbf{z}, \ldots, \mathbf{z}}_{w \text{ times}}) .$$
Note that this is saying that the noise $\mathbf{E}[j,i]$ (i.e., j-th row and i-th column of $\mathbf{E}$) is leaked T times; once in the j-th row of each $\mathsf{ct}_{\mathsf{res}}^{(t)}$.
- After receiving the LWE sample $\mathbf{y} \in \mathcal{R}_q^w$ and leakage $\mathbf{l} \in \mathcal{R}_q^{Tw}$, first split the leakage $\mathbf{l}^T = (\mathbf{l}[1], \ldots, \mathbf{l}[Tw])$ into T separate vectors
$$\mathbf{l}_t^T = (\mathbf{l}[t], \mathbf{l}[T+t], \ldots, \mathbf{l}[(w-1)T+t]) \quad \forall t \in [T] .$$
Note that the vector $\mathbf{l}_t$ will correspond to a leakage of the noise vector $\mathbf{E}[:,i]$ when multiplied with $\mathbf{g}^{-1}(y^{(t)})[i]$, and then hidden by a larger noise $\overline{\mathbf{e}}^{(t)}$.
Then, run and output the result of $\mathcal{D}$ on a simulation of $\mathsf{Hyb}_{3,i}$, with two modifications:
  - the i-th column of ct_1 is computed as $\mathbf{y}^T + \mathbf{m}_1 \cdot g^i$, and
  - for any $t \in [T]$, the ciphertext $\mathsf{ct}_{\mathsf{res}}^{(t)}$ is computed as
$$\mathsf{ct}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + \left(\sum_{i' \in [T] \setminus \{i\}} \mathbf{E}[:,i'] \cdot \mathbf{g}^{-1}(y^{(t)})[i'] + \mathbf{l}_t \right) .$$

The error-leakage Ring-LWE experiment with adversary $\mathcal{A}$ has the following distributions:

- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$. Here, $\mathbf{y}$ corresponds to a real LWE sample, i.e., $\mathbf{y} = \mathbf{a} \cdot \mathbf{s}_1[i] + \mathbf{E}[:,i]$ with leakage $\mathbf{l}_t = \mathbf{E}[:,i] \cdot \mathbf{g}^{-1}(y^{(t)})[i] + \overline{\mathbf{e}}^{(t)}$ (for some fresh key $\mathbf{s}_1[i] \leftarrow\!\$\, \mathcal{R}_q$ and errors $\mathbf{E}[:,i] \leftarrow\!\$\, \mathcal{R}_q^w$ and $\overline{\mathbf{e}}^{(t)} \leftarrow\!\$\, \overline{\chi}^w$).
Therefore, $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{3,i-1}$.
- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$. Here, $\mathbf{y} \leftarrow\!\$\, \mathcal{R}_q^{2m}$ is uniformly random and the leakage is equal to $\mathbf{l}_t = \mathbf{E}[:,i] \cdot \mathbf{g}^{-1}(y^{(t)})[i] + \overline{\mathbf{e}}^{(t)}$ (for fresh errors $\mathbf{E}[:,i] \leftarrow\!\$\, \mathcal{R}_q^w$ and $\overline{\mathbf{e}}^{(t)} \leftarrow\!\$\, \overline{\chi}^w$).
Note that the i-th column of ct_1 will be uniformly random (due to the randomness of $\mathbf{y}$). Therefore, $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{3,i}$.

Therefore, the distinguisher $\mathcal{D}$ has exactly the same advantage as the adversary $\mathcal{A}$. By the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$ assumption, this implies that $\mathsf{Hyb}_{3,i-1} \approx_c^{2^\lambda} \mathsf{Hyb}_{3,i}$.

By the hybrid argument, we get $\mathsf{Hyb}_0 \approx_c^{2^\lambda} \mathsf{Hyb}_4$. Because Hyb_4 is identical to the simulated world, T-times simulation security follows. □

C.1 LHE Parameter Setting

We now prove the claimed efficiencies for the parameter setting in Lemma 6.

Proof (of Lemma 6). Under assumption $\mathsf{LWE}_{\mathcal{R},w,q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$, we may conclude (using Theorem 2 and Lemma 4) that also the error-leakage version $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$ holds, with error distributions χ and $\overline{\chi}$ of parameters $s = 3q^{0.1}$ and $\overline{s} = 2s \cdot g \cdot n\sqrt{T}$.

To obtain the claimed parameters, we choose the gadget-base $g = \lceil q^{0.05} \rceil$ (as used inside of our LEnc construction) and hence $m \leq 20$. Thus, we get

$$\begin{aligned} B_{\mathsf{LHE}} &\leq \Big(\underbrace{g}_{\leq 2q^{0.05}} \cdot \underbrace{m}_{\leq 20} \cdot \underbrace{\gamma_{\mathcal{R}}}_{\leq q^{1/15}} + 2 \cdot \underbrace{g}_{\leq 2q^{0.05}} \cdot \underbrace{n}_{\leq q^{1/15}} \cdot \underbrace{\sqrt{T}}_{\leq q^{1/4}} \Big) \cdot q^{0.1} \cdot \underbrace{\sqrt{\lambda}}_{\leq q^{1/30}} \cdot 3 \\ &\leq 10q^{1/2} \ . \end{aligned}$$

□

D Details on Sel

In this section, we provide the deferred security proof of our batch-select instantiation. In Section D.1, we give the proof for the *random* batch-select version.

Correctness. First, denote by $\mathbf{l} = \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ (over $\mathbb{Z}_p$) be the required output of $\mathsf{Sel.Dec}$, with encoding $\widehat{\mathbf{l}} := \mathsf{Encode}(\mathbf{l})$, which (by the properties of Encode) is equal to $\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2$ over $\mathcal{R}_p$. Considering the larger ring $\mathcal{R}_q$ that our construction works on, we have the identity $\widehat{\mathbf{l}} \cdot \Delta = (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 \cdot \Delta$, because of

$$\begin{aligned} \widehat{\mathbf{l}} \cdot \Delta &= [\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2]_p \cdot \Delta = (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 - p \cdot \mathbf{z}) \cdot \Delta \\ &= (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}}) \cdot \Delta + \widehat{\mathbf{l}}_2 \cdot \Delta - \underbrace{(p \cdot \Delta)}_{=q} \cdot \mathbf{z} \\ &= (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 \cdot \Delta \pmod{q} \ , \end{aligned}$$

where $[.]_p$ denotes reduction modulo p, and $\mathbf{z}$ is some vector in $\mathcal{R}_q^{w'}$. Note that this equality crucially depends on the fact that the plaintext modulus p divides the larger ring modulus q.

Now, by correctness of the underlying LHE, we have

$$\left\| \mathsf{res}' - (\mathbf{r} \cdot d_{\widehat{\mathbf{y}}} + \widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}}) \right\|_\infty \leq B_{\mathsf{LHE}} \ ,$$

and by correctness of the underlying LEnc, we have

$$\left\| \underbrace{\mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}})}_{=:\boldsymbol{\delta}} - (\mathbf{r} \cdot d_{\widehat{\mathbf{y}}} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}) \right\|_\infty \leq B_{\mathsf{LEnc}} \ .$$

Combining these two bounds, the overall error is at most

$$\begin{aligned} \left\| \mathsf{res} - \widehat{\mathbf{l}} \cdot \Delta \right\|_\infty &= \left\| \mathsf{res}' - \boldsymbol{\delta} - ((\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 \cdot \Delta) \right\|_\infty \\ &= \left\| (\mathsf{res}' - \mathbf{r} \cdot d_{\widehat{\mathbf{y}}} - \widehat{\mathbf{l}}_2 \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}) - (\boldsymbol{\delta} - \mathbf{r} \cdot d_{\widehat{\mathbf{y}}} + (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}) + \overline{\mathbf{e}}_{\mathsf{LEnc}} \right\|_\infty \\ &\leq B_{\mathsf{LHE}} + B_{\mathsf{LEnc}} + \|\overline{\mathbf{e}}_{\mathsf{LEnc}}\|_\infty \\ &\leq B_{\mathsf{LHE}} + B_{\mathsf{LEnc}} + \max \overline{\chi}_{\mathsf{LEnc}} < \frac{\Delta}{2} \ , \end{aligned}$$

and therefore the output $\mathsf{Decode}(\lfloor \frac{\mathsf{res}}{\Delta} \rceil)$ will be equal to $\mathsf{Decode}(\widehat{\mathbf{l}}) = \mathbf{l}$.

Lemma 13 (Security of Batch-Select, Construction 3). *Assuming that the underlying* LHE *scheme is* T*-times* 2^λ*-simulation secure, and the underlying* LEnc *scheme is* 2^λ*-simulation secure with* T*-noise leakage, Construction 3 is a correct batch-select scheme that fulfills* T*-times* 2^λ*-simulation security.*

Proof. We define the simulator $\mathsf{Sel.Sim}$, which receives input $\mathsf{pp} = (\mathsf{pp}_{\mathsf{LHE}}, \mathsf{pp}_{\mathsf{LEnc}})$ as well as T message and selection vectors $\{\mathbf{l}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}$, as follows (where we define encoded values as in the construction: $\widehat{\mathbf{y}}^{(t)} := \mathsf{Encode}(\mathbf{y}^{(t)})$ and $\widehat{\mathbf{l}}^{(t)} := \mathsf{Encode}(\mathbf{l}^{(t)})$):

- First, run the laconic encryption simulator

$$\widetilde{\mathsf{LEnc.ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]} \leftarrow \mathsf{LEnc.Sim}(\mathsf{pp}_{\mathsf{LEnc}}, \{\widehat{\mathbf{y}}^{(t)}\}_{[T]}) \ .$$

- Second, compute the digest $d^{(t)}_{\widehat{\mathbf{y}}}$ and evaluate the laconic encryption $\boldsymbol{\delta}^{(t)}$:

$$d^{(t)}_{\widehat{\mathbf{y}}} \leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}}^{(t)}) \quad \forall i \in [T]$$
$$\boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\widetilde{\mathsf{LEnc.ct}}, \widehat{\mathbf{y}}^{(t)}) \quad \forall i \in [T]$$

- Third, simulate the output $\widetilde{\mathsf{res}}'$ of the LHE, and invoke the LHE simulator to simulate all LHE ciphertexts and keys (note that $\boldsymbol{\delta}^{(t)} - \widetilde{\mathbf{e}}^{(t)}$ is equal to the "noise-free" $\mathbf{r} \cdot d^{(t)}_{\widehat{\mathbf{y}}} - ((\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)})$, and therefore $\widetilde{\mathsf{res}}'^{(t)}$ is equal to $\mathbf{r} \cdot d^{(t)}_{\widehat{\mathbf{y}}} + \widehat{\mathbf{l}}^{(t)}_2 \cdot \Delta - \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}$):

$$\overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \leftarrow\!\$\ \overline{\chi}^{w'}_{\mathsf{LEnc}} \quad \forall i \in [T]$$
$$\widetilde{\mathsf{res}}'^{(t)} := \boldsymbol{\delta}^{(t)} - \widetilde{\mathbf{e}}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \quad \forall i \in [T]$$
$$\widetilde{\mathsf{LHE.ct}}_1, \{\widetilde{\mathsf{LHE.ct}}^{(t)}_2, \widetilde{\mathsf{sk}}^{(t)}_{\mathbf{y}^{(t)}}\}_{[T]} \leftarrow \mathsf{LHE.Sim}(\mathsf{pp}, \{\widetilde{\mathsf{res}}'^{(t)}, d^{(t)}_{\widehat{\mathbf{y}}}\}_{[T]})$$

- Finally, return $\widetilde{\mathsf{ct}}_1 := (\widetilde{\mathsf{LEnc.ct}}, \widetilde{\mathsf{LHE.ct}}_1)$ and $\{\widetilde{\mathsf{LHE.ct}}^{(t)}_2, \widetilde{\mathsf{sk}}^{(t)}_{\mathbf{y}^{(t)}}\}_{[T]}$.

We show a series of hybrid experiments that transitions from Hyb_0 (the "real" distribution as defined in Definition 6) to Hyb_3 (the "simulated" distribution). We abuse notation to also write Hyb_i as the output of the distribution of the corresponding experiment.

Hyb_0 To recall, Hyb_0 (omitting generation of public parameters pp) computes $\mathsf{ct}_1 = (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$ and $\{\mathsf{LHE.ct}^{(t)}_2, \mathsf{sk}^{(t)}_{\mathbf{y}^{(t)}}\}_{[T]}$ in the following way (where, as in the construction, we define the digests $d^{(t)}_{\widehat{\mathbf{y}}} := \mathsf{LEnc.Digest}(\widehat{\mathbf{y}}^{(t)})$, and encoded values $\widehat{\mathbf{l}}_1 := \mathsf{Encode}(\mathbf{l}_1)$ and $\widehat{\mathbf{l}}_2 := \mathsf{Encode}(\mathbf{l}_2)$):

$$\mathbf{r}, \boxed{\mathsf{LEnc.ct}} \leftarrow \mathsf{LEnc.Enc}(\widehat{\mathbf{l}}_1 \cdot \Delta)$$
$$\boxed{\mathsf{LHE.ct}_1}, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{r})$$
For $t \in [T]$:
$$\boxed{\mathsf{LHE.ct}^{(t)}_2}, \mathsf{st}^{(t)}_2 \leftarrow \mathsf{LHE.Enc}_2(\widehat{\mathbf{l}}^{(t)}_2 \cdot \Delta - \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}) \quad \Big| \quad \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \leftarrow\!\$\ \overline{\chi}^{w'}_{\mathsf{LEnc}}$$
$$\boxed{\mathsf{sk}^{(t)}_{\mathbf{y}^{(t)}}} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}^{(t)}_2, d^{(t)}_{\widehat{\mathbf{y}}})$$

Hyb_1 We now use the simulation security of LHE to replace generation of $\widetilde{\mathsf{LHE.ct}}_1, \{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]}$ (i.e., the final three lines in Hyb_0) by the following:

$$\widetilde{\mathsf{LHE.ct}}_1, \{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]} \leftarrow \mathsf{LHE.Sim}(\mathsf{pp}, \{\mathsf{res}'^{(t)}, d_{\widehat{\mathbf{y}}}^{(t)}\}_{[T]}) \Big| \; \mathsf{res}'^{(t)} \leftarrow \mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \quad \forall t \in [T]$$

The indistinguishability $\mathsf{Hyb}_0 \approx_c^{2^\lambda} \mathsf{Hyb}_1$ follows directly from the T-times simulation security of LHE.

Hyb_2 In Hyb_2, we now rewrite $\mathsf{res}'^{(t)}$ (for every $t \in [T]$) as

$$\begin{aligned}
\boldsymbol{\delta}^{(t)} &\leftarrow \mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}}^{(t)}) \\
\mathbf{e}^{(t)} &:= \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)}) \\
\mathsf{res}'^{(t)} &\leftarrow \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}
\end{aligned}$$

(Note that $\mathbf{e}^{(t)}$ denotes the LEnc "evaluation error", i.e., the difference between the expected result $\mathbf{r} \cdot y^{(t)} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)}$ and the noisy outcome $\boldsymbol{\delta}^{(t)}$.)

This way of defining $\mathsf{res}'^{(t)}$ is identical to the previous one:

$$\begin{aligned}
\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} &= (\boldsymbol{\delta}^{(t)} + (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)} - \mathbf{e}^{(t)}) + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \\
&= \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \\
&= \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} ,
\end{aligned}$$

where the last equality follows from $\widehat{\mathbf{l}}^{(t)} \cdot \Delta = (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta$, which was shown in the correctness section.

Therefore, we get $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_2$.

Hyb_3 Note that Hyb_2 does not make use of $\widehat{\mathbf{l}}_2^{(t)}$ anymore, but only $\widehat{\mathbf{l}}_1$ (in $\mathsf{LEnc.ct}$ and when computing the LEnc error $\mathbf{e}^{(t)}$). We now use the simulation security of LEnc in order to eliminate the final usages of $\widehat{\mathbf{l}}_1$, by simulating $\mathsf{LEnc.ct}$ and $\mathbf{e}^{(t)}$. Specifically, in Hyb_3, we replace the lines that compute $\mathsf{LEnc.ct}$ and $\mathbf{e}^{(t)}$ by

$$\widetilde{\mathsf{LEnc.ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]} \leftarrow \mathsf{LEnc.Sim}(\mathsf{pp}_{\mathsf{LEnc}}, \{\widehat{\mathbf{y}}^{(t)}\}_{[T]}) .$$

Because $\mathbf{e}^{(t)}$ is only used as part of the quantity $\mathbf{e}^{(t)} + \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}$ (in the definition of $\mathsf{res}'^{(t)}$) with $\overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}$ being a fresh noise generated from $\overline{\chi}_{\mathsf{LEnc}}^{w'}$, we get the indistinguishability $\mathsf{Hyb}_2 \approx_c^{2^\lambda} \mathsf{Hyb}_3$.

Observe that Hyb_3 proceeds identically as the simulator $\mathsf{Sel.Sim}$. By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c^{2^\lambda} \mathsf{Hyb}_3$, which proves the security. □

D.1 *Random* Batch-Select

Lemma 14 (Security of Random Batch-Select, Section 4.4). *Assuming that the underlying* LHE *scheme is* T*-times* 2^λ*-simulation secure, and the underlying* LEnc *scheme is* 2^λ*-simulation secure with* T*-noise leakage, the Random Batch-Select scheme as described in Section 4.4 fulfills* T*-times* 2^λ*-simulation security.*

Proof. To argue security, we describe a simulator Sim^* that first runs the batch-select simulator Sim for our original construction (see proof of Lemma 13), and then additionally simulates the rejection sampling results $d_{i,j}$ and programs the random oracle. It takes as input public parameters pp, messages $\widehat{\mathbf{l}}^{(t)}$, and selection vectors $\mathbf{y}^{(t)}$.

– First, run Sim to obtain

$$(\widetilde{\mathsf{ct}}_1, \{\widetilde{\mathsf{LHE.ct}}_2^{*\,(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}}^{(t)}\}) \leftarrow \mathsf{Sim}(\mathsf{pp}, \{\widehat{\mathbf{l}}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}) \ .$$

– Next, sample a random $\mathsf{seed}^{(t)} \leftarrow \{0,1\}^\lambda$, and simulate the rejection sampling results $d_{i,j}^{(t)}$ as follows. For $t \in [T]$, $i \in [w]$, $j \in [n]$:
- Initially set the number of rejections $d_{i,j}^{(t)} = 0$.
- Sample a noise $e \leftarrow [\Delta]$ and check if

$$|e| < \Delta/2 - (B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}}). \tag{12}$$

- If no, sample a random value $l \leftarrow\!\!\$\ \mathbb{Z}_p$ and program

$$H(\mathsf{seed}^{(t)}, i, j, d_{i,j}^{(t)}) := \widetilde{\mathsf{LHE.ct}}_2^{*\,(t)}[i,j] + l \cdot \Delta + e \bmod q.$$

  Increase $d_{i,j}^{(t)}$ by 1 and repeat.
- If yes, program

$$H(\mathsf{seed}^{(t)}, i, j, d_{i,j}^{(t)}) := \widetilde{\mathsf{LHE.ct}}_2^{*\,(t)}[i,j] + e \bmod q \ .$$

– Finally, output the simulation results $(\widetilde{\mathsf{ct}}_1, \{\widetilde{\mathsf{ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}}^{(t)}\}_{[T]})$, where we choose $\widetilde{\mathsf{ct}}_2^{(t)} := (\mathsf{seed}^{(t)}, (d_{i,j}^{(t)})_{i\in[w],j\in[n]})$.

We show a series of hybrids that transitions from Hyb_0 (the "real" distribution as defined in Definition 8) to Hyb_4 (the "simulated" distribution), focusing on how the rejection results are computed, and how the random oracle is programmed. We abuse the notation to also write Hyb_i as the output distribution of the corresponding experiment.

Hyb_0: We recall how the the rejection results are generated in Hyb_0 (where we surpress the superscript (t) in the following for brevity). First, a vector $\mathbf{c} \in \mathcal{R}_q^w$ is computed as

$$\mathbf{c} = \mathbf{a} \cdot s_2 + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}, \quad s_2 \leftarrow\!\!\$\ \mathcal{R}_q, \quad \overline{\mathbf{e}} \leftarrow \overline{\chi}^w, \quad \overline{\mathbf{e}}_{\mathsf{LEnc}} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^w,$$

where $\mathbf{a}$ is part of the public parameters $\mathsf{Sel.pp}$, and a seed is sampled $\mathsf{seed} \leftarrow \{0,1\}^\lambda$. Finally, an intermediate vector $\mathsf{LHE.ct}_2^*$ is computed as follows. For $i \in [w]$, $j \in [n]$:
– Initially set the number of rejections $d_{i,j} = 0$.
– Check whether Equation 3 is fulfilled (this involves computing $H(\mathsf{seed}, i, j, d_{i,j})$).
– If no, increase $d_{i,j}$ and repeat.
– If yes, set $\mathsf{LHE.ct}_2^*[i,j] = H(\mathsf{seed}, i, j, d_{i,j})$.
Finally, the messages $\widehat{\mathbf{l}}_2$ are computed as $\widehat{\mathbf{l}}_2 = \lfloor(\mathsf{LHE.ct}_2^* - \mathbf{c})/\Delta\rceil \in \mathcal{R}_p^w$.

Hyb_1: Instead of checking Equation 3 for the RO-value $H(\mathsf{seed}, i, j, d_{i,j})$, we do so for a random value $r \leftarrow\!\!\$\ \mathbb{Z}_q$ and then program $H(\mathsf{seed}, i, j, d_{i,j}) := r$. Furthermore, we sample the value r in the following special (but still uniformly random) way:

$$r := \mathbf{c}[i,j] + l \cdot \Delta + e, \quad \text{where } l \leftarrow\!\!\$\ \mathbb{Z}_p \text{ and } e \leftarrow\!\!\$\ [\Delta].$$

We have that $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_0$.

Hyb_2: Note that whether a value r passes the check in Equation 3 only depends on the error e but not l (both defined in the previous step). Hence, instead of checking Equation 3, we will directly check Equation 12 on error e. Further, depending on whether the check is successful, we set r differently:

$$\begin{aligned} &\text{if no:} && r = \mathbf{c}[i,j] + \widehat{\mathbf{l}}_2[i,j] \cdot \Delta + l \cdot \Delta + e, \quad l \leftarrow\!\!\$\, \mathbb{Z}_p \\ &\text{if yes:} && r = \mathbf{c}[i,j] + \widehat{\mathbf{l}}_2[i,j] \cdot \Delta + e, \end{aligned}$$

where $\mathbf{l}_2 \leftarrow \mathcal{R}_q^w$ is a vector sampled only once (for any given t) and reused when a check fails and $d_{i,j}$ is increased.

We again have that $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$ (because in the no-case r looks uniformly random conditioned on not passing Equation 3, and in the yes-case r looks uniformly random conditioned on passing Equation 3).

Hyb_3: In the previous hybrid, we eliminate the generation of $\mathbf{c} \in \mathcal{R}_q^w$ as well as its underlying secret $s_2 \leftarrow\!\!\$\ \mathcal{R}_q$ and errors $\overline{\mathbf{e}}, \overline{\mathbf{e}}_{\mathsf{LEnc}}$. Furthermore, instead of computing r from the term $\mathbf{c}[i,j] + \widehat{\mathbf{l}}_2[i,j] \cdot \Delta$ as in the previous hybrid, we use the term $\mathsf{Sel.ct}_2[i,j]$. These values are generated globally using $(\mathsf{Sel.st}_2, \mathsf{Sel.ct}_2) \leftarrow \mathsf{Sel.Enc}_2(\widehat{\mathbf{l}}_2)$.

Examining the construction of Sel and the underlying construction of LHE, we have that $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4: To summarize, in Hyb_3, the rejection results and programmed entries of the random oracle are entirely derived from $\mathsf{Sel.ct}_2$. Instead of computing $\mathsf{Sel.ct}_2$, we can therefore apply simulation security of the original batch-select scheme. In particular, $\mathsf{Sel.ct}_1$, $\mathsf{Sel.ct}_2$, and $\mathsf{sk}_{\mathbf{y}}$ can all be alternatively generated from $\mathsf{Sel.Sim}(\mathsf{Sel.pp}, \{\widehat{\mathbf{l}}^{(t)}, \mathbf{y}^{(t)}\}_{[T]})$. By Lemma 13, we get $\mathsf{Hyb}_4 \approx_{\mathrm{c}}^{2^\lambda} \mathsf{Hyb}_3$.

Observe that Hyb_4 proceeds identically as the simulated world with simulator Sim^*. By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_{\mathrm{c}}^{2^\lambda} \mathsf{Hyb}_4$, which proves the security. □

D.2 Batch-Select Parameter Settings

We now prove the claimed efficiencies for the parameter setting in Theorem 3 (normal batch-select) and in Theorem 4 (*random* batch-select).

Proof (of Theorem 3). Under the premises of this theorem, both primitives LEnc and LHE underlying our batch-select construction fulfill the error bounds as described in Lemmas 5 and 6. Thus, due to $p \le q^{1/4}$, i.e., $\Delta \ge q^{3/4}$, we get

$$B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} \le \sqrt{q} + 10\sqrt{q} + 1000\sqrt{q} < \frac{1}{2} q^{3/4} \le \Delta/2 \,,$$

and the batch-select construction fulfills correctness (here we used $q \ge 2^{60}$).

The claimed efficienies follow from those of LEnc and LHE: recall that these two building blocks are applied to dimension $w' = \frac{w\ell}{n}$, where $\ell = \lceil \frac{\lambda}{\log p} \rceil$. Due to $\log p = \Omega(\log q)$ and $\log q = O(\lambda)$, this means $\ell = \Theta(\frac{\lambda}{\log q})$. Each ring element has size $n \cdot \log q$, and therefore we get ciphertext sizes

$$\begin{aligned} |\mathsf{ct}_1| &= |\mathsf{LEnc.ct}| + |\mathsf{LHE.ct}_1| = O((w' \log w' + w') \cdot n \cdot \log q) = O(w \log w \cdot \lambda) \quad \text{and} \\ |\mathsf{ct}_2| &= |\mathsf{LHE.ct}_2| = O(w' \cdot n \cdot \lambda) = O(w \cdot \lambda) \,, \end{aligned}$$

and the running time follows in a similar manner.

Weak batch-select with $\frac{\sqrt{q}}{w}$-times simulation security and $|\mathsf{ct}_1| = O(w \cdot \lambda)$ follows with our weak LEnc construction.

A weak batch-select scheme with 1-time simulation security for modulus $q \geq \omega((w\lambda)^2)$ with $|\mathsf{ct}_1| = o(w)$ follows by simply applying the previous weak batch-select scheme in parallel for $k = \omega(\lambda)$ times on smaller instances of size $\frac{w}{k}$, which yields $|\mathsf{ct}_1| = O(w \cdot \lambda/k) = o(w)$ due to $w \leq 2^\lambda$. The new secret key sk will contain k individual keys of size $\mathrm{poly}(\lambda)$ each, and hences its size is still in $\mathrm{poly}(\lambda)$. Note that this trick does not change the size of $|\mathsf{ct}_2|$, because this ciphertext now consists of k individual ciphertexts of size $O(\frac{w}{k} \cdot \lambda)$. □

Proof (of Theorem 4). Under the given parameters (using $q \geq n^{25}$ instead of $q \geq n^{15}$) and $\log w \leq \lambda \leq q^{1/25}$, we get the slightly stronger error bound $B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} \leq O(\frac{q^{1/2}}{\log w})$ in a similar manner as in Lemmas 5 and 6. Therefore, due to $\Delta = q/p \geq q^{1-\epsilon}$, the individual rejection probability is $P \leq O(\frac{q^{1/2}}{\Delta \cdot \log w}) \leq O(\frac{1}{q^{1/2-\epsilon}})$, and the size of $|\mathsf{ct}_2|$ will be, with overwhelming probability, bounded by $O(\frac{w\lambda}{\log q} \cdot \frac{1}{q^{1/2-\epsilon}}) + \mathrm{poly}(\lambda)$.

When $q > 2^{2(1+\varepsilon')\lambda}$ with $\Delta \geq 2^{(2+\varepsilon')\lambda}$ and $p \leq 2^{\varepsilon'\lambda}$, then rejection only happens with probability $\leq O(1/2^\lambda)$, and we may omit ct_2 completely while correctness is satisfied with overwhelming probability. □

E Details on Preprocessing Garbling

In this section, we provide the deferred security proofs of our preprocessing garbling scheme (Construction 4).

Proof (of Lemma 9). We describe the simulator $\mathsf{Sim}_{\mathsf{Priv}}$ required by Definition 14. It takes an offline function U, as well as T online descriptions $\{f^{(t)}\}$ and evaluation results $\{\mathbf{y}^{(t)}\}$. It simulates T garblings and input keys as described below.

– First run the standard model garbling simulator to simulate

$$\widetilde{C}_U^{(t)}, \widetilde{\mathbf{k}}_{\mathbf{f}}^{\mathsf{Fn},(t)}, \widetilde{\mathbf{k}}_{\mathbf{x}}^{(t)} \leftarrow \mathsf{SG.Sim}_{\mathsf{Priv}}(C_U, \mathbf{y}^{(t)}) \quad \forall t \in [T] \,.$$

– Next, simulate the batch-select output message vector randomly as

$$\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)} \leftarrow\!\!\$\ \mathcal{M}^{s_U} \,,$$

and apply T-times security of Sel to simulate

$$\mathsf{Sel.pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}),$$
$$\widetilde{\mathsf{Sel.ct}}_1, \{\widetilde{\mathsf{Sel.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)}\} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}, \{\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}, \mathbf{f}^{(t)}\}).$$

– Finally, simulate the key translation ciphertexts in accordance with the batch-select output $\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}$: for the ciphertexts $\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i}$ that are decodable by the evaluator, we choose

$$\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} := H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i]) + \widetilde{\mathbf{k}}_{\mathbf{f}}^{\mathsf{Fn},(t)}[i] \,,$$

and we sample the remaining ciphertexts uniformly at random:

$$\widetilde{\mathsf{ct}}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i} \leftarrow\!\!\$\ \mathcal{K} \,.$$

Output the simulated reusable garbling part $\widehat{U}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \widetilde{\mathsf{Sel.ct}_1})$, and for each iteration $t \in [T]$ the simulated offline garbling $\widehat{U}^{(t)} = (\widetilde{C}_U^{(t)}, \widetilde{\mathsf{Sel.ct}_2}^{(t)}, \{\widetilde{\mathsf{ct}}_{0,i}^{\prime(t)}, \widetilde{\mathsf{ct}}_{1,i}^{\prime(t)}\})$, online garbling $\mathsf{hint}^{(t)} = \widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)}$, and input keys $\widetilde{\mathbf{k}}_{\mathbf{x}}^{(t)}$.

We show a series of hybrids that transitions from $\mathsf{Hyb}_0 = \mathsf{Exp}_{\mathsf{Priv}}^{\mathcal{A},0}(\lambda)$ to $\mathsf{Hyb}_2 = \mathsf{Exp}_{\mathsf{Priv}}^{\mathcal{A},1}(\lambda)$. We abuse the notation to also write Hyb_i as the output distribution of the experiment.

Hyb_0: We briefly recall the game $\mathsf{Exp}_{\mathsf{Priv}}^{\mathcal{A},0}(\lambda)$. $\mathcal{A}$ selects $1^{\ell_x}, 1^{\ell_y}$, an offline function U, and T online descriptions $f^{(t)} \in \mathcal{F}_U$ (with bit representations $\mathbf{f}^{(t)} \in \{0,1\}^{s_U}$) and inputs $\mathbf{x}^{(t)} \in \{0,1\}^{\ell_x}$. The information received by the adversary is indicated through boxed terms.

$$\boxed{\mathsf{Sel.pp}} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}) \quad \Big| \quad \mathbf{l}_1 \leftarrow\!\$\ \mathcal{M}^{s_U} \tag{13}$$
$$\boxed{\mathsf{Sel.ct}_1}, \mathsf{Sel.st}_1 \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1)$$

for $t \in [T]$:

$$\boxed{\mathsf{Sel.ct}_2^{(t)}}, \mathsf{Sel.st}_2^{(t)} \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2^{(t)}) \quad \Big| \quad \mathbf{l}_2^{(t)} \leftarrow\!\$\ \mathcal{M}^{s_U} \tag{14}$$
$$\boxed{\mathsf{sk}_{\mathbf{f}^{(t)}}^{(t)}} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{Sel.st}_1, \mathsf{Sel.st}_2^{(t)}, \mathbf{f}^{(t)})$$

$$\boxed{\widehat{C}_U^{(t)}} \leftarrow \mathsf{SG.Garble}(C_U, (\mathbf{K}^{\mathsf{Fn},(t)}, \mathbf{K}^{(t)})) \quad \Big| \quad \mathbf{K}^{\mathsf{Fn},(t)} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{s_U}) \tag{15}$$
$$\boxed{\mathbf{k}_{\mathbf{x}}^{(t)}} = \mathbf{K}^{(t)}[\mathbf{x}^{(t)}]. \qquad \mathbf{K}^{(t)} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{\ell_x})$$

$$\boxed{\mathsf{ct}_{0,i}^{\prime(t)}} := H(\mathbf{l}_2^{(t)}[i]) + \mathbf{K}^{\mathsf{Fn},(t)}[i,0] \tag{16}$$
$$\boxed{\mathsf{ct}_{1,i}^{\prime(t)}} := H(\mathbf{l}_1[i] + \mathbf{l}_2^{(t)}[i]) + \mathbf{K}^{\mathsf{Fn},(t)}[i,1]$$

Hyb_1: Instead of computing $\mathsf{Sel.pp}, \mathsf{Sel.ct}_1$ and $\{\mathsf{Sel.ct}_2^{(t)}, \mathsf{sk}_{\mathbf{f}}^{(t)}\}$ as above (Equation 13 and 14), Hyb_1 simulates them using $\mathsf{Sel.Sim}$:

$$\boxed{\mathsf{Sel.pp}} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}) \quad \Big| \quad \mathbf{l}_1 \leftarrow\!\$\ \mathcal{M}^{s_U} \tag{17}$$
$$\boxed{\widetilde{\mathsf{Sel.ct}}_1, \{\widetilde{\mathsf{Sel.ct}_2}^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)}\}} \quad \mathbf{l}_2^{(t)} \leftarrow\!\$\ \mathcal{M}^{s_U}$$
$$\leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}, \{\mathbf{l}_{\mathsf{res}}^{(t)}, \mathbf{f}^{(t)}\}). \quad \mathbf{l}_{\mathsf{res}}^{(t)} := \mathbf{l}_1 \odot \mathbf{f}^{(t)} + \mathbf{l}_2^{(t)}$$

The T-times simulation security of Sel guarantees that the (boxed) simulated terms are indistinguishable from the correctly computed ones. We have $|\Pr[\mathsf{Hyb}_1(\lambda) = 1] - \Pr[\mathsf{Hyb}_0(\lambda)] = 1| \leq \mathsf{negl}(\lambda)$.

Hyb_2: Instead of generating $\mathbf{l}_2^{(t)}$ and then computing $\mathbf{l}_{\mathsf{res}}^{(t)}$ from it, we change Equation 17 in such a way that it generates both $\mathbf{l}_1 \leftarrow\!\$\ \mathcal{M}^{s_U}$ and $\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)} \leftarrow\!\$\ \mathcal{M}^{s_U}$ uniformly random. Then, we modify Equation 16 s.t. for any $t \in [T]$ and $i \in [s_U]$, the key translation ciphertexts are computed as

$$\boxed{\widetilde{\mathsf{ct}}_{\mathbf{f}^{(t)}[i],i}^{\prime(t)}} := \mathbf{k}_{\mathbf{f}}^{\mathsf{Fn},(t)} + H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i]) \quad \Big| \quad \mathbf{k}_{\mathbf{f}}^{\mathsf{Fn},(t)} := \mathbf{K}^{\mathsf{Fn},(t)}[i, \mathbf{f}^{(t)}[i]] \tag{18}$$

$$\boxed{\mathsf{ct}_{1-\mathbf{f}^{(t)}[i],i}^{\prime(t)}} := \mathbf{K}^{\mathsf{Fn},(t)}[i, 1-\mathbf{f}^{(t)}[i]] + \begin{cases} H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i] + \mathbf{l}_1[i]) & \text{if } \mathbf{f}^{(t)}[i] = 0 \\ H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i] - \mathbf{l}_1[i]) & \text{if } \mathbf{f}^{(t)}[i] = 1 \end{cases} \tag{19}$$

Note that the distribution is unchanged, and we have $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$.

Hyb_3: Now we can apply the correlation-robustness of H: Instead of computing the "unused" key translation ciphertexts as in Equation 19, we simulate it uniformly random:

$$\boxed{\widetilde{\mathsf{ct}}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i}} \leftarrow\!\$\ \mathcal{K}$$

By correlation-robustness of H (which may be applied because $\mathbf{l}_1 \leftarrow\!\$\ \mathcal{M}^{s_U}$ is used nowhere but in the definition of Equation 19), the second summand in the definition of $\mathsf{ct}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i}$ can be replaced by uniformly random. This allows us to equivalently replace $\mathsf{ct}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i}$ itself by uniformly random, and therefore we have $|\Pr[\mathsf{Hyb}_3(\lambda)=1] - \Pr[\mathsf{Hyb}_2(\lambda)]=1| \le \mathsf{negl}(\lambda)$.

Hyb_4: Instead of computing $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn},(t)}$ as in Equation 18 and $\widehat{C}_U^{(t)}, \mathbf{k}_{\mathbf{x}}^{(t)}$ as in Equation 15, Hyb_4 simulates them using $\mathsf{SG.Sim}_{\mathsf{Priv}}$:

$$\boxed{\{\widetilde{C}_U^{(t)}, \widetilde{\mathbf{k}}_{\mathbf{f}}^{\mathsf{Fn},(t)}, \widetilde{\mathbf{k}}_{\mathbf{x}}^{(t)}\}} \leftarrow \mathsf{SG.Sim}_{\mathsf{Priv}}(\{C_U, \mathbf{y}^{(t)}\}_{t\in[T]}). \Bigg| \mathbf{y}^{(t)} = U(f^{(t)}, \mathbf{x}^{(t)})$$

By input privacy (Definition 15 simplified for standard garbling, i.e., $T=1$ and without RDGen and $\mathsf{GarbleFunc}$), this hybrid is indistinguishable from the previous one: $|\Pr[\mathsf{Hyb}_4(\lambda)=1] - \Pr[\mathsf{Hyb}_3(\lambda)]=1| \le \mathsf{negl}(\lambda)$.

By a hybrid argument, we conclude that $|\Pr[\mathsf{Hyb}_4(\lambda)=1] - \Pr[\mathsf{Hyb}_0(\lambda)]=1| \le \mathsf{negl}(\lambda)$, which proves the theorem. □

F Adaptive Security and Malicious 2PC

The purpose of this section is to formulate a stronger adaptive security for batch-select (Definition 20) and show that our batch-select scheme (Construction 3), unmodified, is adaptively secure if the two ingredients $\mathsf{LEnc}, \mathsf{LHE}$ satisfy approprate adaptive security notions (see Sections F.1 and F.2 for definitions and modified constructions). We then sketch how an adaptively secure batch select scheme (Section F.3), together with an adaptive standard garbling scheme, results in an adaptive preprocess garbling scheme (Section F.4).

Finally, in Section F.5 we apply the adaptively secure batch select scheme to construct a T-session malicious 2PC protocol with a function independent preprocessing phase, and T *succinct* function dependent preprocessing phases. Our construction is a natural application of batch select to the authenticated garbling framework [WRK17, KRRW18, DILO22, CWYY23] for malicious 2PC.

F.1 Adaptive Security of LEnc

Definition 18 (Adaptive T-noise Leakage Simulation Security of LEnc). *A* LEnc *scheme is* adaptive T-noise leakage simulation secure *w.r.t. a noise distribution* $\overline{\chi}_{\mathsf{LEnc}}$ *if there exist two efficient simulators* $\mathsf{LEnc.Sim}_1, \mathsf{LEnc.Sim}_2$ *such that for any efficient adversary* $\mathcal{A}$*,*

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

where the game $\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{LEnc}}(\lambda)$ *is defined as follows.*

1. $\mathcal{A}(1^\lambda)$ *decides a message dimension* 1^w*, and receives* $\mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w)$.
2. $\mathcal{A}$ *decides a message vector* $\mathbf{s} \in \mathcal{R}_q^w$ *and receives* ct*, where*

$$\begin{aligned} &\textit{If } b = 0 && (\mathbf{r}, \mathsf{ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{s}) \\ &\textit{If } b = 1 && (\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{LEnc.Sim}_1(\mathsf{pp}). \end{aligned}$$

3. *Repeate the following for* $t = 1, \ldots, T$*. In the end* $\mathcal{A}$ *outputs a bit* b' *as the outcome of the game.*
   - $\mathcal{A}$ *decides a database* $\mathbf{a}^{(t)} \in \mathcal{R}_q^w$ *and receives a leakage* $\mathbf{e}^{(t)} + \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}$*, where*

$$\begin{aligned} &\overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \leftarrow \overline{\chi}^w && \boldsymbol{\delta} = \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)}) \quad d^{(t)}_{\mathbf{a}} = \mathsf{LEnc.Digest}(\mathbf{a}^{(t)}) \\ &\textit{If } b = 0 && \mathbf{e}^{(t)} = \boldsymbol{\delta} - \mathbf{r} \cdot d^{(t)}_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a} \\ &\textit{If } b = 1 && \mathbf{e}^{(t)} \leftarrow \mathsf{LEnc.Sim}_2(\mathsf{st}, \mathbf{a}^{(t)}). \end{aligned}$$

Our original LEnc construction already fulfills adaptivity, assuming the stronger *adaptive* error-leakage Ring-LWE assumption:

Lemma 15 (Adaptive security of Construction 1). *Assuming* adaptive $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi}_{\mathsf{LEnc}},T,\mathcal{L}_{1,2m}(g)}$*, Construction 1 (*$\mathsf{LEnc}$*) fulfills* adaptive simulation security with T-noise leakage *(Definition 18).*

Proof. The proof is similar to that of Lemma 10. We naturally split the simulator (previously Sim) into two separate simulators Sim_1 and Sim_2 in the following way:

- $\mathsf{Sim}_1(\mathsf{pp})$ samples all ciphertexts purely random

$$\widetilde{\mathbf{C}}_i \leftarrow\!\!\$\ \mathcal{R}_q^{w \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1\ ,$$

  and outputs ciphertext $\widetilde{\mathsf{ct}} := (\widetilde{\mathbf{C}}_0, \ldots, \widetilde{\mathbf{C}}_{\ell-1})$ and state $\mathsf{st} := \mathsf{pp}$.
- $\mathsf{Sim}_2(\mathsf{st}, \mathbf{a}^{(t)})$ simulates the noise leakages as follows: sample

$$\mathbf{E}_i \leftarrow\!\!\$\ \chi^{w \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1\ ,$$

  and for each $\mathsf{ind} \in \{0,1\}^\ell$, compute

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}:i}\ .$$

  Return $\widetilde{\mathbf{e}}^{(t)}$.

The hybrids are the same as before, except that instead of $\mathsf{Hyb}_{i^*,j}$ outputting the values $((\mathbf{b}_0, \mathbf{b}_1), (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}), \{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}\}_{[T]})$, we turn it into an interactive game $\mathsf{Hyb}^{\mathcal{D}}_{i^*,j}$ with an adversary $\mathcal{D}$. The adversary first receives $(\mathbf{b}_0, \mathbf{b}_1)$ as in Equation 5, then chooses a message vector $\mathbf{s}$ and receives ciphertexts $(\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$ as in Equation 6, and finally (for each $t \in [T]$) chooses a database $\mathbf{a}^{(t)}$ and receives $\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}$ as in Equation 7. In the end it outputs a decision bit.

We now show that $|\Pr[\mathsf{Hyb}^{\mathcal{D}}_{i^*,j} = 1] - \Pr[\mathsf{Hyb}^{\mathcal{D}}_{i^*,j+1} = 1]|$ is negligible for any $i^* \in [\ell]$ and $0 \le j < w$. To do so, we construct an adversary $\mathcal{A}$ for $\mathsf{a\text{-}elLWE}$ as follows.

– First, $\mathcal{A}$ receives public parameters $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$ and a Ring-LWE sample $\mathbf{y} \in \mathcal{R}_q^{2m}$. Pass $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$ on to $\mathcal{D}$ who outputs message vector $\mathbf{s} \in \mathcal{R}_q^w$.
Then, compute ciphertexts $\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}$ as in $\mathsf{Hyb}_{i^*,j}$, except for the ind-th row of $\mathbf{C}_{i^*}$:

$$\mathbf{C}_{i^*}[\mathsf{ind}] := \mathbf{y}^T + \mathbf{r}_{i^*+1}[\mathsf{ind}] \cdot \left(\overline{\mathsf{ind}_{i^*}} \cdot \mathbf{g}^T \; \mathsf{ind}_{i^*} \cdot \mathbf{g}^T\right)$$

Send ciphertexts $\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}$ to $\mathcal{D}$.
– For each $t \in [T]$: $\mathcal{D}$ outputs a database $\mathbf{a}^{(t)}$. Use these to compute vectors $\mathbf{z}_{\mathsf{pre}}^{(t)}$ for any prefix $\mathsf{pre} \in \{0,1\}^\ell$.
Submit the leakage matrix $(\mathbf{z}_{\mathsf{ind},:i^*}^{(t)})^T$ to obtain leakage $\mathbf{l}[t] \in \mathcal{R}_q$. Compute errors $\widetilde{\mathbf{e}}_{\mathsf{res}}^{(t)}$ as in $\mathsf{Hyb}_{i^*,j}$, except for

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{\substack{0 \le i < \ell \\ i \neq i^*}} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind},i}^{(t)} \quad \text{and} \quad \widetilde{\mathbf{e}}_{\mathsf{res}}^{(t)}[\mathsf{ind}] := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \mathbf{l}[t] \; .$$

Send $\widetilde{\mathbf{e}}_{\mathsf{res}}^{(t)}$ to $\mathcal{D}$.
– Output the same decision bit as $\mathcal{D}$.

As in the proof of Lemma 10, we get $\mathsf{a\text{-}elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}_{1,2m}(g)} \equiv \mathsf{Hyb}^{\mathcal{D}}_{i^*,j}$ and $\mathsf{a\text{-}elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}_{1,2m}(g)} \equiv \mathsf{Hyb}^{\mathcal{D}}_{i^*,j+1}$. Thus, by $\mathsf{a\text{-}elLWE}$, we get $\mathsf{Hyb}^{\mathcal{D}}_{i^*,j} \approx_{\mathrm{c}} \mathsf{Hyb}^{\mathcal{D}}_{i^*,j+1}$.

By adaptive leakage-error Ring-LWE and the hybrid argument, we get $\mathsf{Hyb}_0 \approx_{\mathrm{c}} \mathsf{Hyb}_\ell$, which concludes the security proof. □

F.2 Adaptive Security of LHE

Definition 19 (Adaptive T-times Simulation Security of LHE). *An* LHE *scheme is* adaptive T-times simulation secure *if there exist three efficient simulators* $\mathsf{LHE.Sim}_1, \mathsf{LHE.Sim}_2, \mathsf{LHE.Sim}_3$ *such that for any efficient adversary* $\mathcal{A}$*,*

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

where the game $\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{LHE}}(\lambda)$ *is defined as follows.*

1. $\mathcal{A}(1^\lambda)$ *decides a message dimension* 1^w*, and receives* $\mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w)$.
2. $\mathcal{A}$ *decides the first message vector* $\mathbf{l}_1 \in \mathcal{R}_q^w$ *and receives* ct_1*, where*

If $b = 0$	$(\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{LHE.Enc}_1(\mathbf{m}_1)$
If $b = 1$	$(\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{LHE.Sim}_1(\mathsf{pp})$.

3. *Repeated the following for* $t = 1, \ldots, T$*. In the end* $\mathcal{A}$ *outputs a bit* b' *as the outcome of the game.*
   – $\mathcal{A}$ *decides a second message vector* $\mathbf{m}_2^{(t)} \in \mathcal{R}_q^w$ *and receives* ct_2*, where*

If $b = 0$	$(\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{LHE.Enc}_2(\mathbf{m}_2)$
If $b = 1$	$(\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{LHE.Sim}_2(\mathsf{st}_1)$.

– $\mathcal{A}$ *decides an element* $y \in \mathcal{R}_q$ *and receives* $\mathsf{sk}_y^{(t)}$*, where*

$$\begin{aligned} &\textit{If } b = 0 \qquad \mathsf{sk}_y^{(t)} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, y^{(t)}) \\ &\textit{If } b = 1 \qquad \mathsf{sk}_y^{(t)} \leftarrow \mathsf{LHE.Sim}_3(\mathsf{st}_2^{(t)}, y^{(t)}, \mathbf{m}_{\mathsf{res}}^{(t)}), \quad \mathbf{m}_{\mathsf{res}}^{(t)} = \mathbf{m}_1 \odot y^{(t)} + \mathbf{m}_2^{(t)}. \end{aligned}$$

We need to modify our original construction of LHE and add a random oracle H in order to obtain adaptivity:

Construction 6 (Adaptive LHE over rings LHE).

$\underline{\mathsf{Setup}(1^\lambda, 1^w) \to \mathsf{pp}}$**:** Output a public random matrix $\mathsf{pp} = \mathbf{a} \leftarrow\!\!\$\, \mathcal{R}_q^w$.

$\underline{\mathsf{Enc}_1(\mathbf{m}_1) \to \mathsf{ct}_1, \mathsf{st}_1}$**:** Sample Ring-LWE secrets $\mathbf{s}_1 \leftarrow\!\!\$\, \mathcal{R}_q^m$ and *truncated* noises $\mathbf{E} \leftarrow\!\!\$\, \overline{\mathcal{D}}_{\mathcal{R},s}^{w \times m}$, Output a ciphertext

$$\mathsf{ct}_1 := \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \in \mathcal{R}_q^{w \times m} ,$$

together with state $\mathsf{st}_1 = \mathbf{s}_1$.

$\underline{\mathsf{Enc}_2(\mathbf{m}_2) \to \mathsf{ct}_2, \mathsf{st}_2}$**:** Sample a Ring-LWE secret $s_2 \leftarrow\!\!\$\, \mathcal{R}_q$ and *truncated* noises $\overline{\mathbf{e}} \leftarrow\!\!\$\, \overline{\mathcal{D}}_{\mathcal{R},\bar{s}}^{w}$. Generate $r \leftarrow\!\!\$\, \{0,1\}^\lambda$. Output a ciphertext

$$\mathsf{ct}_2 := \mathbf{a} \cdot s_2 + \mathbf{m}_2 + \overline{\mathbf{e}} + H(r) \quad \in \mathcal{R}_q^w ,$$

together with state $\mathsf{st}_2 = (\mathbf{s}_2, r)$.

$\underline{\mathsf{KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y) \to \mathsf{sk}'_y}$**:** Parse the states $\mathsf{st}_1 = \mathbf{s}_1 \in \mathcal{R}_q^m$ and $\mathsf{st}_2 = (s_2 \in \mathcal{R}_q, r \in \{0,1\}^\lambda)$. Output $\mathsf{sk}'_y := (\mathsf{sk}_y, r)$, where sk_y is the decryption key

$$\mathsf{sk}_y := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2 \quad \in \mathcal{R}_q .$$

$\underline{\mathsf{Dec}((\mathsf{sk}_y, r), \mathsf{ct}_1, \mathsf{ct}_2, y) \to \mathbf{m}_{\mathsf{res}}}$**:** First combine the ciphertexts into

$$\begin{aligned} \mathsf{ct}_{\mathsf{res}} &:= \mathsf{ct}_1 \cdot \mathbf{g}^{-1}(y) + \mathsf{ct}_2 - H(r) \quad \in \mathcal{R}_q^w , \\ \text{// s.t. } \mathsf{ct}_{\mathsf{res}} &= \mathbf{a} \cdot (\mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2) + (\mathbf{m}_1 \cdot \mathbf{g}^T \cdot \mathbf{g}^{-1}(y) + \mathbf{m}_2) + \text{noise} \end{aligned}$$

Then, use sk_y to decrypt and return $\mathbf{m}_{\mathsf{res}} = \mathsf{ct}_{\mathsf{res}} - \mathbf{a} \cdot \mathsf{sk}_y$, which is supposed to equal $\mathbf{m}_{\mathsf{res}} = \mathbf{m}_1 \cdot y + \mathbf{m}_2 + \text{noise}$.

Lemma 16 (Security of Construction 6). *Assuming* $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$*, Construction 2 (adaptive LHE) fulfills* adaptive T-times simulation security *(Definition 19) in the Random Oracle Model.*

Proof. The proof is analogous to that of Lemma 10. The main difference is that we need to additionally handle the term $H(r)$ used to mask the ciphertext ct_2.

The simulator manages the random oracle H. Whenever a query is being made (including queries made by the adversary), the simulator answers consistently, sampling random elements upon each query that was never seen before.

– $\mathsf{Sim}_1(\mathbf{a})$ samples ct_1 uniformly random:

$$\widetilde{\mathsf{ct}}_1 \leftarrow\!\!\$\, \mathcal{R}_q^{w \times m}$$

– $\mathsf{Sim}_2(\mathsf{st}_1)$ samples $\mathsf{ct}_2^{(t)}$ uniformly random:

$$\widetilde{\mathsf{ct}}_2^{(t)} \leftarrow\!\!\$\ \mathcal{R}_q^w$$

– $\mathsf{Sim}_3(\mathsf{st}_2^{(t)}, y^{(t)}, \mathbf{m}_{\mathsf{res}}^{(t)})$ first samples the key uniformly random:

$$\widetilde{\mathsf{sk}}_y^{(t)} \leftarrow\!\!\$\ \mathcal{R}_q$$

Then, it programs the random oracle by choosing $r^{(t)} \leftarrow\!\!\$\ \{0,1\}^\lambda$ and setting

$$\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \quad \forall t \in [T]$$
$$H(r^{(t)}) := \mathsf{ct}_2^{(t)} - (\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)}))\ .$$

Return $\widetilde{\mathsf{sk}'_y}^{(t)} := (\widetilde{\mathsf{sk}}_y^{(t)}, r^{(t)})$.

Hyb_0 To recall, Hyb_0 generates ciphertexts $\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}\}$ and decryption keys $\{\mathsf{sk}_y^{(t)}\}$ in the following way:

$$\boxed{\mathbf{a}} \leftarrow\!\!\$\ \mathcal{R}_q^m \tag{20}$$

$$\boxed{\mathsf{ct}_1} := \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \Big| \quad \begin{array}{l} \mathbf{s}_1 \leftarrow\!\!\$\ \mathcal{R}_q^m \\ \mathbf{E} \leftarrow\!\!\$\ \overline{\mathcal{D}}_{\mathcal{R},s}^{w\times m} \end{array} \tag{21}$$

For $t \in [T]$:

$$\boxed{\mathsf{ct}_2^{(t)}} := (\mathbf{a} \cdot s_2^{(t)} + \mathbf{m}_2^{(t)} + \overline{\mathbf{e}}^{(t)}) + H(r^{(t)}) \quad \Big| \quad \begin{array}{l} s_2^{(t)} \leftarrow\!\!\$\ \mathcal{R}_q \\ \overline{\mathbf{e}}^{(t)} \leftarrow\!\!\$\ \overline{\mathcal{D}}_{\mathcal{R},\bar{s}}^{w} \\ r^{(t)} \leftarrow\!\!\$\ \{0,1\}^\lambda \end{array} \tag{22}$$

$$\boxed{\mathsf{sk}_y^{(t)}} := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}, \boxed{r^{(t)}} \tag{23}$$

Hyb_1 As a first step, in Hyb_1 we sample the errors $\mathbf{E}$ and $\overline{\mathbf{e}}^{(t)}$ as Gaussians from $\chi^{w\times m} = \mathcal{D}_{\mathcal{R},s}^{w\times m}$ and $\overline{\chi}^w = \mathcal{D}_{\mathcal{R},\bar{s}}^{w}$, instead of truncated Gaussians from $\overline{\mathcal{D}}_{\mathcal{R},s}^{w\times m}$ and $\overline{\mathcal{D}}_{\mathcal{R},\bar{s}}^{w}$, respectively. By Lemma 2, we have $\mathsf{Hyb}_1 \approx_{\mathrm{s}} \mathsf{Hyb}_0$.

Hyb_2 Now, we defer choosing the "actual" value of $\mathsf{ct}_2^{(t)}$ (i.e., its value after removing the mask $H(r^{(t)})$) until the adversary has chosen $y^{(t)}$. In particular, we sample $\mathsf{ct}_2^{(t)}$ uniformly at random, and before revealing $r^{(t)}$, we program the random oracle H on input r in such a way that $\mathsf{ct}_2^{(t)} - H(r^{(t)})$ has the value $\mathbf{a} \cdot s_2^{(t)} + \mathbf{m}_2^{(t)} + \overline{\mathbf{e}}^{(t)}$ as before. Formally,

$$\boxed{\mathsf{ct}_2^{(t)}} \leftarrow\!\!\$\ \mathcal{R}_q^w$$
$$\boxed{\mathsf{sk}_y^{(t)}} := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}, \boxed{r^{(t)}} \quad \Big| \quad \begin{array}{l} s_2^{(t)} \leftarrow\!\!\$\ \mathcal{R}_q \\ r^{(t)} \leftarrow\!\!\$\ \{0,1\}^\lambda \end{array}$$

Furthermore, before sending $\mathsf{sk}_y^{(t)}$ and $r^{(t)}$ to the adversary, $H(r^{(t)})$ is programmed as follows:

$$H(r^{(t)}) := \mathsf{ct}_2^{(t)} - (\mathbf{a} \cdot s_2^{(t)} + \mathbf{m}_2^{(t)} + \overline{\mathbf{e}}^{(t)}) \quad \Big| \quad \overline{\mathbf{e}}^{(t)} \leftarrow\!\!\$\ \overline{\chi}^w$$

Note that this hybrid Hyb_2 is statistically close to Hyb_1: conditioned on the adversary $\mathcal{A}$ not querying $H(r^{(t)})$ before the programming step is completed, the two worlds are identical. Furthermore, the adversary $\mathcal{A}$ will not query $H(r^{(t)})$ with more than negligible probability, because $r^{(t)}$ is freshly sampled right before programming $H(r^{(t)})$.

Hyb_3 Now we continue in a similar way as in the proof of Lemma 12: instead of computing $H(r^{(t)})$ directly from $\mathbf{m}_2^{(t)}$, we first define the *combined* ciphertext $\mathsf{ct}_{\mathsf{res}}^{(t)}$, and then simulate $H(r^{(t)})$ as a combination of $\mathsf{ct}_{\mathsf{res}}^{(t)}$ and ct_1.

More, precisely, $H(r^{(t)})$ is programmed as follows:

$$H(r^{(t)}) := \mathsf{ct}_2^{(t)} - (\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)})) \;\Bigg|\; \begin{array}{l} \overline{\mathbf{e}}^{(t)} \leftarrow\!\$\; \overline{\chi}^w \\ \widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \end{array}$$

By definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$ and $\mathbf{m}_{\mathsf{res}}^{(t)} = \mathbf{m}_1 \cdot y^{(t)} + \mathbf{m}_2^{(t)}$, this way of choosing $H(r^{(t)})$ is identical to the previous one. Therefore, we have $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 Note that in the previous hybrid, the uniformly random key $s_2^{(t)}$ is used nowhere but in the definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$. Therefore, in Hyb_2, we may equivalently generate

$$\boxed{\widetilde{\mathsf{sk}}_y^{(t)}} \leftarrow\!\$\; \mathcal{R}_q, \boxed{r^{(t)}} \leftarrow\!\$\; \{0,1\}^\lambda$$

uniformly random instead of computing $\widetilde{\mathsf{sk}}_y^{(t)}$ from $s_2^{(t)}$. We get $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_3$.

Hyb_5 In this hybrid, we replace the first ciphertext $\mathsf{ct}_1 = \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E}$ by a uniformly generated vector

$$\widetilde{\mathsf{ct}}_1 \leftarrow\!\$\; \mathcal{R}_q^{w \times m} .$$

We define sub-hybrids $\mathsf{Hyb}_{4,i}$ for $i = 0, \ldots, m$, where $\mathsf{Hyb}_{4,i}$ is identical to Hyb_4, except that the first i columns of ct_1 are sampled uniformly random. Note that $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_{4,0}$ and $\mathsf{Hyb}_{4,m} \equiv \mathsf{Hyb}_5$. Therefore, in order to prove $\mathsf{Hyb}_5 \approx_{\mathrm{c}} \mathsf{Hyb}_4$, it only remains to show $\mathsf{Hyb}_{3,i-1} \approx_{\mathrm{c}} \mathsf{Hyb}_{3,i}$ for $i = 1, \ldots, m$. To do so, assume there exists a distinguisher $\mathcal{D}$ for $\mathsf{Hyb}_{4,i-1}$ and $\mathsf{Hyb}_{4,i}$. We construct an adversary $\mathcal{A}$ for $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}_{1,1}^{\times w}(g)}$ in the following way (where H-queries by $\mathcal{D}$ are handled the same way as by our simulator):

- First, $\mathcal{A}$ receives public parameters $\mathbf{a} \in \mathcal{R}_q^w$ and a Ring-LWE sample $\mathbf{y} \in \mathcal{R}_q^w$. Pass $\mathsf{pp} := \mathbf{a}$ on to $\mathcal{D}$, who outputs the first message vector $\mathbf{m}_1 \in \mathcal{R}_q^w$.
  Then, compute the first ciphertext ct_1 as in $\mathsf{Hyb}_{4,i-1}$, except for the i-th column, which is chosen to be $\mathbf{y}$. Send ct_1 to $\mathcal{D}$.
- For each $t \in [T]$: $\mathcal{D}$ outputs the second message vector $\mathbf{m}_2^{(t)} \in \mathcal{R}_q^w$.
  Then, we sample $\mathsf{ct}_2^{(t)} \leftarrow\!\$\; \mathcal{R}_q^w$ randomly and send $\mathsf{ct}_2^{(t)}$ to $\mathcal{D}$, who outputs the element $y^{(t)} \in \mathcal{R}_q$.
  Now, submit the leakage matrix

  $$\mathrm{diag}(\mathbf{g}^{-1}(y^{(t)})[i], \ldots, \mathbf{g}^{-1}(y^{(t)})[i])$$

  to the challenger, who outputs leakage $\mathbf{l}_t$ (*this is saying that* $\mathbf{l}_t$ *is a leakage of the noise* $\mathbf{E}[:, i]$ *when multiplied with* $\mathbf{g}^{-1}(y^{(t)})[i]$).
  Then, sample $\mathsf{sk}_y^{(t)} \leftarrow\!\$\; \mathcal{R}_q$ and $r^{(t)} \leftarrow\!\$\; \{0,1\}^\lambda$ randomly (as in Hyb_4), and program

  $$H(r^{(t)}) := \mathsf{ct}_2^{(t)} - (\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)})) ,$$

where $\mathsf{ct}_{\mathsf{res}}^{(t)}$ is computed as

$$\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + \left(\sum_{i' \in [m] \setminus \{i\}} \mathbf{E}[:, i'] \cdot \mathbf{g}^{-1}(y^{(t)})[i'] + \mathbf{l}_t \right) .$$

Send $(\mathsf{sk}_y^{(t)}, r)$ to $\mathcal{D}$.

– Output the same decision bit as $\mathcal{D}$.

As in the proof of Lemma 12, we get $\mathsf{a\text{-}elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)} \equiv \mathsf{Hyb}^{\mathcal{D}}_{4,i-1}$ and $\mathsf{a\text{-}elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)} \equiv \mathsf{Hyb}^{\mathcal{D}}_{4,i}$. Thus, by $\mathsf{a\text{-}elLWE}$, we get $\mathsf{Hyb}^{\mathcal{D}}_{4,i-1} \approx_{\mathrm{c}} \mathsf{Hyb}^{\mathcal{D}}_{4,i}$.

By the hybrid argument, we get $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_5$. Because Hyb_5 is identical to the simulated world, adaptive T-times simulation security follows. □

F.3 Adaptive Security of Batch-Select

We now formalize adaptive T-times simulation security of the batch-select scheme below. In contrast to the selective version (Definition 6), where the challenge messages $\mathbf{l}_1, \mathbf{l}_2^{(1)}, \ldots, \mathbf{l}_2^{(T)}$ and selection vectors $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(T)}$ are decided in one-shot, we now allow an adversary to adaptively query messages and selection vectors, and receive corresponding ciphertexts and decryption keys immediately.

Definition 20 (Adaptive T-times Simulation Security of Sel). *A Sel scheme is* adaptive T-times simulation secure *if there exist three efficient simulators $\mathsf{Sel.Sim}_1, \mathsf{Sel.Sim}_2, \mathsf{Sel.Sim}_3$ such that for any efficient adversary $\mathcal{A}$,*

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

where the game $\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{Sel}}(\lambda)$ is defined as follows.

1. *$\mathcal{A}(1^\lambda)$ decides a batch size 1^w, and receives $\mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w)$.*
2. *$\mathcal{A}$ decides the first message vector $\mathbf{l}_1 \in \mathcal{M}^w$ and receives ct_1, where*

$$\begin{aligned} &\textit{If } b = 0 && (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ &\textit{If } b = 1 && (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Sim}_1(\mathsf{pp}). \end{aligned}$$

3. *Repeated the following for $t = 1, \ldots, T$. In the end $\mathcal{A}$ outputs a bit b' as the outcome of the game.*
   – *$\mathcal{A}$ decides a second message vector $\mathbf{l}_2^{(t)} \in \mathcal{M}^w$ and receives ct_2, where*

$$\begin{aligned} &\textit{If } b = 0 && (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2) \\ &\textit{If } b = 1 && (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Sim}_2(\mathsf{st}_1). \end{aligned}$$

   – *$\mathcal{A}$ decides a selection vector $\mathbf{y}^{(t)} \in \{0,1\}^w$ and receives $\mathsf{sk}_{\mathbf{y}}^{(t)}$, where*

$$\begin{aligned} &\textit{If } b = 0 && \mathsf{sk}_{\mathbf{y}}^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, \mathbf{y}^{(t)}) \\ &\textit{If } b = 1 && \mathsf{sk}_{\mathbf{y}}^{(t)} \leftarrow \mathsf{Sel.Sim}_3(\mathsf{st}_2^{(t)}, \mathbf{y}^{(t)}, \mathbf{l}_{\mathsf{res}}^{(t)}), \quad \mathbf{l}_{\mathsf{res}}^{(t)} = \mathbf{l}_1 \odot \mathbf{y}^{(t)} + \mathbf{l}_2^{(t)}. \end{aligned}$$

It turns out that our original construction, unmodified, is already adaptively secure if the two ingradients $\mathsf{LEnc}, \mathsf{LHE}$ satisfy their respective adaptive security notions (see Appendices F.1, and F.2).

The proof of the following lemma is analogous to Lemma 13 of selective security. We include the proof in Appendix D for completeness.

Lemma 17 (Adaptive Security of Construction 3). *Assuming that the underlying* LHE *scheme is* adaptively T*-times simulation secure, and the underlying* LEnc *scheme is* adaptively *simulation secure with* T*-noise leakage, our earlier construction for batch select (Construction 3) fulfills* adaptive T*-times simulation security.*

Proof. The simulator is similar to that of Lemma 13, but we need to split it into the following three parts:

- $\mathsf{Sel.Sim}_1(\mathsf{pp})$: Run

$$\widetilde{\mathsf{LEnc.ct}}, \mathsf{LEnc.st} \leftarrow \mathsf{LEnc.Sim}_1(\mathsf{pp}_{\mathsf{LEnc}})$$
$$\widetilde{\mathsf{LHE.ct}}_1, \mathsf{LHE.st}_1 \leftarrow \mathsf{LHE.Sim}_1(\mathsf{pp}_{\mathsf{LHE}})$$

  and output ciphertext $\widetilde{\mathsf{ct}}_1 := (\widetilde{\mathsf{LEnc.ct}}, \widetilde{\mathsf{LHE.ct}}_1)$ and state $\mathsf{st}_1 = (\mathsf{LEnc.st}, \mathsf{LHE.st}_1)$.
- $\mathsf{Sel.Sim}_2(\mathsf{st}_1)$: Run

$$\widetilde{\mathsf{LHE.ct}}_2, \mathsf{LHE.st}_2 \leftarrow \mathsf{LHE.Sim}_2(\mathsf{LHE.st}_1)$$

  and output ciphertext $\widetilde{\mathsf{ct}}_2 := \widetilde{\mathsf{LHE.ct}}_2$ and state $\mathsf{st}_2 = (\mathsf{LEnc.st}, \mathsf{LHE.st}_2)$.
- $\mathsf{Sel.Sim}_3(\mathsf{st}_2, \mathbf{y}, \mathbf{l})$: Compute the encoded values $\widehat{\mathbf{y}} := \mathsf{Encode}(\mathbf{y})$ and $\widehat{\mathbf{l}} := \mathsf{Encode}(\mathbf{l})$. Simulate the LEnc noise and evaluate the laconic encryption $\boldsymbol{\delta}$:

$$\widetilde{\mathbf{e}} \leftarrow \mathsf{LEnc.Sim}_2(\mathsf{LEnc.st}, \widehat{\mathbf{y}})$$
$$\boldsymbol{\delta} \leftarrow \mathsf{LEnc.Eval}(\widetilde{\mathsf{LEnc.ct}}, \widehat{\mathbf{y}})$$

  Then, simulate the output $\widetilde{\mathsf{res}}'$ of the LHE and accordingly its key $\widetilde{\mathsf{sk}}_{\mathbf{y}}$ as follows, where the digest is computed as $d_{\widehat{\mathbf{y}}} \leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}})$:

$$\overline{\mathbf{e}} \leftarrow\!\!\$\ \overline{\chi}_{\mathsf{LEnc}}^{w'}$$
$$\widetilde{\mathsf{res}}' := \boldsymbol{\delta} - \widetilde{\mathbf{e}} + \widehat{\mathbf{l}} \cdot \Delta - \overline{\mathbf{e}}$$
$$\widetilde{\mathsf{sk}}_{\mathbf{y}} \leftarrow \mathsf{LHE.Sim}_3(\mathsf{LHE.st}_2, \widetilde{\mathsf{res}}', d_{\widehat{\mathbf{y}}})$$

  Output $\widetilde{\mathsf{sk}}_{\mathbf{y}}$.

We use an analogous series of hybrid experiments that transitions from Hyb_0 (the "real" distribution as defined in Definition 6) to Hyb_3 (the "simulated" distribution). We abuse notation to also write Hyb_i as the output of the distribution of the corresponding experiment.

Hyb_0 To recall, Hyb_0 (omitting generation of public parameters pp) computes in the first phase $\mathsf{ct}_1 = (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$, and in the second phase, for each $t \in [T]$, (1) $\mathsf{LHE.ct}_2^{(t)}$, and

(2) $\mathsf{sk}^{(t)}_{\mathbf{y}^{(t)}}$, in the following way (where, as in the construction, we define the digests $d_{\widehat{\mathbf{y}}} := \mathsf{LEnc.Digest}(\widehat{\mathbf{y}}^{(t)})$, and encoded values $\widehat{\mathbf{l}}_1 := \mathsf{Encode}(\mathbf{l}_1)$ and $\widehat{\mathbf{l}}_2 := \mathsf{Encode}(\mathbf{l}_2)$):

$$\mathbf{r}, \boxed{\mathsf{LEnc.ct}} \leftarrow \mathsf{LEnc.Enc}(\widehat{\mathbf{l}}_1 \cdot \Delta)$$
$$\boxed{\mathsf{LHE.ct}_1}, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{r})$$
For $t \in [T]$:
$$\boxed{\mathsf{LHE.ct}_2^{(t)}}, \mathsf{sk}_2^{(t)} \leftarrow \mathsf{LHE.Enc}_2(\widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}) \quad \Big| \quad \overline{\mathbf{e}}^{(t)} \leftarrow\!\!\$\ \overline{\chi}^{w'}_{\mathsf{LEnc}}$$
$$\boxed{\mathsf{sk}^{(t)}_{\mathbf{y}^{(t)}}} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{sk}_1, \mathsf{sk}_2^{(t)}, d^{(t)}_{\widehat{\mathbf{y}}})$$

Hyb_1 We now use the *adaptive* simulation security of LHE to replace generation of $\widetilde{\mathsf{LHE.ct}}_1$, $\{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}^{(t)}_{\mathbf{y}^{(t)}}\}_{[T]}$ (i.e., the final three lines in Hyb_0) by the following:

$$\boxed{\widetilde{\mathsf{LHE.ct}}_1}, \mathsf{LHE.st}_1 \leftarrow \mathsf{LHE.Sim}_1(\mathsf{LHE.pp})$$
For $t \in [T]$:
$$\boxed{\widetilde{\mathsf{LHE.ct}}_2^{(t)}}, \mathsf{LHE.st}_2^{(t)} \leftarrow \mathsf{LHE.Sim}_2(\mathsf{LHE.st}_1)$$
$$\boxed{\widetilde{\mathsf{sk}}^{(t)}_{\mathbf{y}^{(t)}}} \leftarrow \mathsf{LHE.Sim}(\mathsf{LHE.st}_2^{(t)}, d^{(t)}_{\widehat{\mathbf{y}}}, \mathsf{res}'^{(t)}) \quad \Big| \quad \begin{array}{l} \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \leftarrow\!\!\$\ \overline{\chi}^{w'}_{\mathsf{LEnc}} \\ \mathsf{res}'^{(t)} \leftarrow \mathbf{r} \cdot d^{(t)}_{\widehat{\mathbf{y}}} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \end{array}$$

The indistinguishability $\mathsf{Hyb}_0 \approx_{\mathrm{c}} \mathsf{Hyb}_1$ follows directly from the adaptive T-times simulation security of LHE.

Hyb_2 In Hyb_2, we now rewrite $\mathsf{res}'^{(t)}$ as

$$\boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}}^{(t)}) \quad \forall i \in [T]$$
$$\mathbf{e}^{(t)} := \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot d^{(t)}_{\widehat{\mathbf{y}}} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)})$$
$$\mathsf{res}'^{(t)} \leftarrow \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \quad \forall t \in [T]$$

Note that $\mathbf{e}^{(t)}$ denotes the LEnc "evaluation error", i.e., the difference between the expected result $\mathbf{r} \cdot d^{(t)}_{\widehat{\mathbf{y}}} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)}$ and the noisy outcome $\boldsymbol{\delta}^{(t)}$.
As in the proof of Lemma 13, we have $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_2$.

Hyb_3 Note that Hyb_2 does not make use of $\widehat{\mathbf{l}}_2^{(t)}$ anymore, but only $\widehat{\mathbf{l}}_1$ (in $\mathsf{LEnc.ct}$ and when computing the LEnc error $\mathbf{e}^{(t)}$). We now use the simulation security of LEnc in order to eliminate the final usages of $\widehat{\mathbf{l}}_1$, by simulating $\mathsf{LEnc.ct}$ and $\mathbf{e}^{(t)}$. Specifically, in Hyb_3, we replace the computation of $\mathbf{r}$ and $\mathsf{LEnc.ct}$ by the simulation

$$\mathsf{LEnc.st}, \widetilde{\mathsf{LEnc.ct}} \leftarrow \mathsf{LEnc.Sim}_1(\mathsf{pp}_{\mathsf{LEnc}}) \ ,$$

and the computation of the noise $\mathbf{e}^{(t)}$ by the simulation

$$\widetilde{\mathbf{e}}^{(t)} \leftarrow \mathsf{LEnc.Sim}_2(\mathsf{LEnc.st}, \widehat{\mathbf{y}}^{(t)})$$

Because $\mathbf{e}^{(t)}$ is only used as part of the quantity $\mathbf{e}^{(t)} + \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}$ (in the definition of $\mathsf{res}'^{(t)}$) with $\overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}$ being a fresh noise generated from $\overline{\chi}^{w'}$, we get the indistinguishability $\mathsf{Hyb}_2 \approx_{\mathrm{c}} \mathsf{Hyb}_3$.

Observe that Hyb_3 proceeds identically as the simulated world. By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$, which proves the security. □

F.4 Adaptive Security of Preprocessing Garbling

In this section, we sketch a simple way of achieving *adaptive* T-times input privacy for preprocessing garbling in the RO model, assuming a batch-select that fulfills adaptive security as above, and an adaptively secure garbling scheme in the standard model.

First, we need to modify the preprocessing garbling interface (Definition 14) slightly: $\mathsf{GarbleFunc}(\mathsf{st}, f) \to \mathsf{hint}, d$ will output some output decoding information d in addition to the hint. Furthermore, $\mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{k}_{\mathbf{x}}, d)$ utilizes this decoding information for evaluating the circuit.

Defining Adaptive T-times Input Privacy. In contrast to standard T-times input privacy (Definition 15), the adversary can now choose the online descriptions $f^{(t)}$ and inputs $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{x}}}$ one after another:

1. The adversary only selects $1^{\ell_{\mathsf{x}}}, 1^{\ell_{\mathsf{y}}}$ and the offline function $U \in \mathcal{U}_{\ell_{\mathsf{x}},\ell_{\mathsf{y}}}$. It receives $\widehat{U}_{\mathsf{rd}}$ and $\{\widehat{U}^{(t)}\}$, generated in the real resp. simulated world as follows:

$$\begin{aligned}(\widehat{U}_{\mathsf{rd}}, \mathsf{st}_{\mathsf{rd}}) &\leftarrow \mathsf{RDGen}(U)\\ \mathbf{K}^{(t)} &\leftarrow \mathsf{InputKeyGen}(1^\lambda, 1^{\ell_{\mathsf{x}}}) \quad \text{or} \quad \widehat{U}_{\mathsf{rd}}, \{\widehat{U}^{(t)}\} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}(U)\ .\\ (\widehat{U}^{(t)}, \mathsf{st}^{(t)}) &\leftarrow \mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K}^{(t)})\end{aligned}$$

2. Then, for each $t \in [T]$:
   - The adversary chooses the online description $f^{(t)}$, and receives the $\mathsf{hint}^{(t)}$, generated in the real resp. simulated world as follows:

$$\mathsf{hint}^{(t)}, d^{(t)} \leftarrow \mathsf{GarbleFunc}(\mathsf{st}^{(t)}, f^{(t)}) \quad \text{or} \quad \mathsf{hint}^{(t)} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}(f^{(t)})\ .$$

   - The adversary chooses the input $\mathbf{x}^{(t)} \in \{0,1\}^{\ell_{\mathsf{x}}}$, and receives the input labels $\mathbf{k}_{\mathbf{x}}^{(t)}$ and input decoding d generated in the real resp. simulated world as follows:

$$\mathbf{k}_{\mathbf{x}}^{(t)} = \mathbf{K}^{(t)}[\mathbf{x}^{(t)}] \quad \text{or} \quad \mathbf{k}_{\mathbf{x}}^{(t)}, d^{(t)} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}(U(f^{(t)}, \mathbf{x}^{(t)}))\ .$$

For a *standard* garbling scheme (without separate preprocessing phase or T-reusability) to fulfill adaptive input privacy, the definition above is simplified towards $T = 1$ and skipping the second stage (since the online description $f^{(t)}$ does not contain any information).

Modifying the Preprocessing Garbling Scheme. In the adaptive security game above, note that with our plain preprocessing garbling scheme (Construction 4), the adversary would be able to view the function's input labels $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}$ to the garbling $\widehat{C}_U$ already after selecting the online description $f^{(t)}$. However, it may adaptively choose the input $\mathbf{x}^{(t)}$ after this step. Therefore, we cannot apply (adaptive) SG input privacy to show our scheme secure.

Instead, we will modify Construction 4 slightly. We account for the issue above by hiding $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}$ until the adversary has received the decoding information d in the final step. Specifically, we require the hash function H used for key translation (which previously just needed to fulfill correlation-robustness) to be a Random Oracle, and modify the scheme's algorithms in the following way.

– In $\mathsf{GarbleU}$, we generate an additional $\mathsf{seed} \leftarrow\!\!\$\ \mathcal{M}^{s_U}$, which is used for computing the key translation ciphertexts

$$\mathsf{ct}'_{0,i} := H(\mathbf{l}_2[i] \ + \mathsf{seed}) + \mathbf{K}^{\mathsf{Fn}}[i,0]$$
$$\mathsf{ct}'_{1,i} := H(\mathbf{l}_1[i] + \mathbf{l}_2[i] \ + \mathsf{seed}) + \mathbf{K}^{\mathsf{Fn}}[i,1]$$

– $\mathsf{GarbleFunc}$ additionally outputs the decoding information, which is exactly the seed: $d := \mathsf{seed}$.
– Eval replaces its computation of the function's input labels by

$$\mathbf{k}^{\mathsf{Fn}}_{\mathbf{f}}[i] := \mathsf{ct}'_{\mathbf{f}[i],i} - H(\mathbf{l}_{\mathsf{res}}[i] \ + \mathsf{seed}) \ .$$

Proving Adaptive T-times Input Privacy. In the security game for the scheme described above, the function's input labels $\mathbf{k}^{\mathsf{Fn}}_{\mathbf{f}}$ to the garbling $\widehat{C}_U$ of the universal circuit are effectively hidden by seed until the adversary also receives the remaining input labels $\mathbf{k}_{\mathbf{x}}$.

We now sketch how to show security, assuming that the underlying scheme SG satisfies adaptive privacy, the batch-select scheme Sel satisfies adaptive security, and H is a programmable random oracle.

The three stages of our simulator will be as follows:

– $\mathsf{Sim}_{\mathsf{Priv}}(U)$ simulates the reusable part of the garbling as $\widehat{U}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \mathsf{Sel.ct}_1)$, where

$$\mathsf{Sel.pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}), \quad (\mathsf{Sel.st}_1, \widetilde{\mathsf{Sel.ct}_1}) \leftarrow \mathsf{Sel.Sim}_1(\mathsf{Sel.pp}) \ .$$

It also simulates the non-reusable offline garblings $\widehat{U}^{(t)} := (\widetilde{C}^{(t)}_U, \widetilde{\mathsf{Sel.ct}_2}, \{\widetilde{\mathsf{ct}}'^{(t)}_{0,i}, \widetilde{\mathsf{ct}}'^{(t)}_{1,i}\})$, where

$$\widehat{C}^{(t)}_U \leftarrow \mathsf{Sim}^{\mathsf{SG}}_{\mathsf{Priv}}.\mathsf{Sim}(C_U), \quad (\mathsf{Sel.st}^{(t)}_2, \widetilde{\mathsf{Sel.ct}_2}^{(t)}) \leftarrow \mathsf{Sel.Sim}_2(\mathsf{Sel.st}_1), \quad \widetilde{\mathsf{ct}}'^{(t)}_{0,i}, \widetilde{\mathsf{ct}}'^{(t)}_{1,i} \leftarrow\!\!\$\ \mathcal{K}$$

– $\mathsf{Sim}_{\mathsf{Priv}}(f^{(t)})$ then uses the final batch-select simulator to simulate the $\mathsf{hint} := \widetilde{\mathsf{sk}}^{(t)}_{\mathbf{f}^{(t)}}$ as

$$\widetilde{\mathsf{sk}}^{(t)}_{\mathbf{f}^{(t)}} \leftarrow \mathsf{Sel.Sim}_3(\mathsf{Sel.st}^{(t)}_2, \mathbf{f}^{(t)}, \widetilde{\mathbf{l}_{\mathsf{res}}}^{(t)}) \ ,$$

where $\widetilde{\mathbf{l}_{\mathsf{res}}}^{(t)} \leftarrow\!\!\$\ \mathcal{M}^{s_U}$ is uniformly random.
– $\mathsf{Sim}_{\mathsf{Priv}}(\mathbf{y})$ then runs the standard garbling simulator to obtain the input keys

$$\mathbf{k}^{\mathsf{Fn},(t)}_{\mathbf{f}^{(t)}}, \mathbf{k}^{(t)}_{\mathbf{x}} \leftarrow \mathsf{Sim}^{\mathsf{SG}}_{\mathsf{Priv}}.\mathsf{Sim}(\mathbf{y}) \ .$$

It programs the random oracle to "fix" the decodable key translation ciphertext:

$$H(\widetilde{\mathbf{l}}^{(t)}_{\mathsf{res}}[i] + \mathsf{seed}) := \mathsf{ct}'^{(t)}_{\mathbf{f}^{\mathsf{Fn},(t)}[i],i} - \mathbf{k}^{(t)}_{\mathbf{f}^{(t)}}[i] \quad \forall i \in [s_U]$$

for some random $\mathsf{seed} \leftarrow\!\!\$\ \mathcal{M}^{s_U}$. Then, it outputs input keys $\mathbf{k}^{(t)}_{\mathbf{x}}$ and decoding information $d := \mathsf{seed}$.

The hybrids are very similar to that in the proof of Lemma 9.

– First, starting from the real world, we replace the outputs of the batch-select scheme Sel by its simulations. This eliminates the need to know messages $\mathbf{l}_1$ and $\mathbf{l}^{(t)}_2$: simulation of $\mathsf{Sel.ct}_1$ and $\mathsf{Sel.ct}_2$ happens in the first stage of the game, and simulation of $\mathsf{sk}^{(t)}_{\mathbf{f}^{(t)}}$ happens in the second stage, for which only the "visible" messages $\mathbf{l}_{\mathsf{res}} := \mathbf{l}_1 \odot \mathbf{f}^{(t)} + \mathbf{l}^{(t)}_2$ are required.

– Second, we also modify generation of the key translation ciphertexts, in such a way that it only depends on a uniformly random $\mathbf{l}_{\mathsf{res}} \leftarrow\!\!\$\ \mathcal{M}^{s_U}$ (there is no usage of individual messages $\mathbf{l}_1$ and $\mathbf{l}_2^{(t)}$ anymore):

$$\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} := H(\widetilde{\mathbf{l}}^{(t)}_{\mathsf{res}}[i] + \mathsf{seed}) + \widetilde{\mathbf{k}}^{\mathsf{Fn},(t)}_{\mathbf{f}}[i]$$
$$\widetilde{\mathsf{ct}}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i} \leftarrow\!\!\$\ \mathcal{K}$$

We can do so because H is a random oracle, and therefore it is unlikely that the adversary will ever query H on two inputs that differ in exactly $\mathbf{l}_1$.

– Now we can use the RO properties to delay choosing the key translation ciphertexts until the final phase where the output $\mathbf{y}$ is known: we generate $\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} \leftarrow\!\!\$\ \mathcal{K}$ uniformly random, and in the final phase we program

$$H(\widetilde{\mathbf{l}}^{(t)}_{\mathsf{res}}[i] + \mathsf{seed}) := \widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} - \widetilde{\mathbf{k}}^{\mathsf{Fn},(t)}_{\mathbf{f}}[i]\ .$$

This is indistinguishable from the previous hybrid, because seed is completely hidden from the adversary, so it is unlikely that it will query $\widetilde{\mathbf{l}}^{(t)}_{\mathsf{res}}[i] + \mathsf{seed}$ before this programming step.

– Finally, the function's input labels $\mathbf{k}^{\mathsf{Fn},(t)}_{\mathbf{f}^{(t)}}$ are not used until the final stage (where the output $\mathbf{y}$ is known) to program H. Therefore, we may use adaptive input privacy of the garbling of the universal circuit to replace $\widehat{C}^{(t)}_U$ (computed in the first stage) and input labels $\mathbf{k}^{\mathsf{Fn},(t)}_{\mathbf{f}^{(t)}}, \mathbf{k}^{(t)}_{\mathbf{x}}$ (computed in the final stage) by

$$\widehat{C}^{(t)}_U \leftarrow \mathsf{Sim}^{\mathsf{SG}}_{\mathsf{Priv}}.\mathsf{Sim}(C_U)$$
$$\mathbf{k}^{\mathsf{Fn},(t)}_{\mathbf{f}^{(t)}}, \mathbf{k}^{(t)}_{\mathbf{x}} \leftarrow \mathsf{Sim}^{\mathsf{SG}}_{\mathsf{Priv}}.\mathsf{Sim}(\mathbf{y})$$

F.5 Preprocessing T-Session Malicious 2PC

In this section, we construct a protocol, in the random oracle model, realizing the T-session 2PC functionality (Figure 1) against malicious adversaries. The protocol has a instance-independent preprocessing phase, where Alice and Bob know only an upper bound s_U of the function discription length, with amortized $O(\lambda \cdot s_U)$ bits of communication. In each session, it has a succinct function dependent preprocessing phase with $\mathrm{poly}(\lambda)$ bits of communication, and an online phase with with $\ell_{\mathsf{x}A} + \ell_{\mathsf{x}B} + \mathrm{poly}(\lambda)$ bits of communication.

We will first recall and summarize the authenticated garbling framework of [WRK17, DILO22] for malicious 2PC protocols, and then describe how to achieve succinct function dependent and online phases with our *adaptively* secure batch select scheme in the framework.

The Authenticated Garbling Framework. The framework has three main steps, which we summarize below. While the original framework considers evaluating a target circuit C with two private inputs $\mathbf{x}_A, \mathbf{x}_B$ from Alice and Bob, it can be easily extended to support circuits with an additional *public* input $\mathbf{x}_P$ that's only known at the input step together with $\mathbf{x}_A, \mathbf{x}_B$.

Preprocessing: Alice and Bob jointly run a sub-protocol $\mathsf{AuthGarb}^C$ (Figure 3), parameterized by the target circuit C.

– Alice obtains input keys $\mathbf{K}_P, \mathbf{K}_A, \mathbf{K}_B$, an input mask $\mathbf{a}_I$, and decryption information D.
– Bob obtains a garbled circuit $\overline{C}$ and an input mask $\mathbf{b}_I$.

Input:

Functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$

For $t = 1, \ldots, T$:

- Upon receiving $(t, f_A^{(t)}, \mathbf{x}_A^{(t)})$ from Alice or $(t, f_B^{(t)}, \mathbf{x}_B^{(t)})$ from Bob, leak $f_A^{(t)}$ or $f_B^{(t)}$ to the adversary.
- After receiving inputs from both Alice and Bob:
  1. If $f_A^{(t)}$ or $f_B^{(t)}$ has description length more than s_U, or if $f_A^{(t)} \neq f_B^{(t)}$, output $\perp$ to Bob and abort.
  2. Otherwise, output $\emptyset$ to Alice and $(t, f_A^{(t)}(\mathbf{x}_A^{(t)}, \mathbf{x}_B^{(t)}))$ to Bob.

Fig. 1: The T-session 2PC functionality.

1. Bob masks his input $\overline{\mathbf{x}}_B = \mathbf{x}_B \oplus \mathbf{b}_I$, and sends $\overline{\mathbf{x}}_B$ to Alice.
2. Alice masks her input $\overline{\mathbf{x}}_A = \mathbf{x}_A \oplus \mathbf{a}_I$ and selects input labels according to the inputs, $\mathbf{k}_P = \mathbf{K}_P[\mathbf{x}_P]$, $\mathbf{k}_A = \mathbf{K}_A[\overline{\mathbf{x}}_A]$ $\mathbf{k}_B = \mathbf{K}_B[\overline{\mathbf{x}}_B]$. Alice sends $D, \mathbf{k}_P, \mathbf{k}_A, \mathbf{k}_B, \overline{\mathbf{x}}_A$ to Bob.

Output: Bob locally runs an evaluation algorithm $\mathsf{AuthEval}$ (Figure 4) and outputs the result $\mathbf{y} \leftarrow \mathsf{AuthEval}^C(\overline{C}, D, \mathbf{k}_P, \mathbf{k}_A, \mathbf{k}_B, \mathbf{x}_P, \overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$.

For completeness, we recreate the sub-protocol $\mathsf{AuthGarb}^C$ and the evaluation algorithm $\mathsf{AuthEval}^C$ in Figures 3 and 4.

In [DILO22], the authors showed that the preprocessing step, i.e. the $\mathsf{AuthGarb}$ sub-protocol, can be realized in constant rounds and with $O(|C| \cdot (\lambda + \kappa))$ bits of communication using pseudorandom correlation generators (PCG) for vector oblivious linear evaluation [BCGI18, CRR21] (VOLE) and multiplication triples [BCG+20] (MT) correlations. We refer to [DILO22] for more details.

Minimizing Function Dependent Communication Using Batch Select. The idea is analoguous to how we minimize the function dependent garbling size in our preprocessing garbling construction.

In an instance *independent* offline phase, Alice and Bob run the $\mathsf{AuthGarb}$ sub-protocol on a universal circuit U that takes a function description f (of bounded length s_U) as the public input, and two inputs $\mathbf{x}_A$, $\mathbf{x}_B$ from Alice and Bob.

In the function *dependent* offline phase, it remains for Alice to transmit the input labels selected by the target function descriptoin f to Bob. Our batch select scheme lets Alice achieve this succinctly, with $\mathrm{poly}(\lambda)$ bits. Applying the same idea for the input labels selected by Alice's and Bob's inputs also reduces the online phase communication to $\mathrm{poly}(\lambda)$ bits.

We illustrate the modified protocol below. For simplicity, we assume here a batch select scheme Sel with matching message space to encrypt the input labels.

Instance Independent Offline:

1. Alice and Bob jointly run the sub-protocol $\mathsf{AuthGarb}^U$, w.r.t. a universal circuit U.
   - Alice gets input keys $\mathbf{K}_P, \mathbf{K}_A, \mathbf{K}_B$, an input mask $\mathbf{a}_I$, and decryption information D.
   - Bob gets a garbled circuit $\overline{U}$ and an input mask $\mathbf{b}_I$.
2. Alice encrypts the input keys using batch select as follows, and the decryption information using a random oracle $\mathsf{ct}_O = D \oplus H(\mathsf{seed})$,

$$(\mathsf{st}_{P,1}, \mathsf{Sel.ct}_{P,1}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_P[\mathbf{1}] - \mathbf{K}_P[\mathbf{0}]), \quad (\mathsf{st}_{P,2}, \mathsf{Sel.ct}_{P,2}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_P[\mathbf{0}]),$$
$$(\mathsf{st}_{I,1}, \mathsf{Sel.ct}_{I,1}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_I[\mathbf{1}] - \mathbf{K}_I[\mathbf{0}]), \quad (\mathsf{st}_{I,2}, \mathsf{Sel.ct}_{I,2}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_I[\mathbf{0}]),$$

where $\mathbf{K}_I := \mathbf{K}_A \,\|\, \mathbf{K}_B$, and $\mathsf{seed} \leftarrow \{0,1\}^\lambda$. Alice sends the ciphertexts $\mathsf{Sel.ct}_{P,1}$, $\mathsf{Sel.ct}_{P,2}$, $\mathsf{Sel.ct}_{I,1}$, $\mathsf{Sel.ct}_{I,2}$, ct_O to Bob.

Function Dependent Offline:

1. Alice computes and sends a decryption key $\mathsf{sk}_P \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{P,1}, \mathsf{st}_{P,2}, f)$ to Bob.
2. Bob locally decrypts input labels for f, $\mathbf{k}_P \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_P, \mathsf{Sel.ct}_{P,1}, \mathsf{Sel.ct}_{P,2}, f)$.

Online:

1. Bob masks his masked input $\overline{\mathbf{x}}_B = \mathbf{x}_B \oplus \mathbf{b}_I$ to Alice.
2. Alice masks her input $\overline{\mathbf{x}}_A = \mathbf{x}_A \oplus \mathbf{a}_I$ and computes a decryption key accordingly: $\mathsf{sk}_I \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{I,1}, \mathsf{st}_{I,2}, \overline{\mathbf{x}}_A \,\|\, \overline{\mathbf{x}}_B)$. Alice sends $\overline{\mathbf{x}}_A$, sk_I, seed to Bob.
3. Bob locally decrypts input labels for $\mathbf{k}_A, \mathbf{k}_B$, output information D, and runs the evaluation algorithm to recover the result as in the original framework.

Amortizing the Instance Independent Offline Phase. While achieving succinct instance dependent and online phases, the above solution incurs an overhead to the instance independent phase. First, Alice and Bob runs the sub-protocol $\mathsf{AuthGarb}$ on a universal circuit U, which is at least $O(\log |f|)$ times larger than the actual target circuit f. Second, the first batch select ciphertext $\mathsf{Sel.ct}_{P,1}$ is concretely much larger than the authenticated garbling size of the universal circuit $\overline{U}$. (The cost of sending $\mathsf{Sel.ct}_{P,2}$ and the public parameters $\mathsf{Sel.pp}$ are concretely similar to sending $\overline{U}$.)

Fortunately our batch select scheme allows re-using $\mathsf{Sel.ct}_{P,1}$ upto T times. We leverage this reusability to amortize the communication overhead of sending $\mathsf{Sel.ct}_{P,1}$, and construct a T-session 2PC protocol. When $T = \Omega(\log |f|)$, the amortized function independent preprocessing takes $O(\lambda \cdot |f| \log |f|) = O(\lambda \cdot s_U)$ bits of communication.

We describe our T-session 2PC protocol in Figure 2. The ingradients to the protocol are:

- the $\mathsf{AuthGarb}$ sub-protocol (Figure 3), and the evaluation algorithm $\mathsf{AuthEval}$ (Figure 4) from the authenticated garbling framework.
- a batch select scheme Sel with message space $\mathcal{M}$ and adaptive T-times simulation security;
- two random oracles $H : \{0,1\}^\lambda \to \{0,1\}^*$, $H' : \mathcal{M} \to \{0,1\}^\lambda$.

The communication costs of the protocol are:

- $O(T \cdot \lambda \cdot s_U)$ bits in the instance independent offline phase;
- $\mathrm{poly}(\lambda)$ in each function dependent offline phase;
- $\ell_{\mathsf{x}A} + \ell_{\mathsf{x}B} + \mathrm{poly}(\lambda)$ bits in each online phase.

Theorem 5. *Let λ be the computational security parameter, and $s_U = \mathrm{poly}(\lambda)$ be an upper bound on target circuits' description length. Assuming the underlying batch select scheme Sel has adaptive T-times simulation security, the protocol $\mathsf{2PC}^{T,s_U}$ UC-realizes the T-session 2PC functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$ in the $\mathcal{F}_{\mathsf{pre}}$-hybrid model and in the random oracel (RO) model, the presence of malicious adversaries who statically corrupt one of the participants.*

Before proving Theorem 5, we briefly recap the UC framework.

Overview of the Universal Composibility (UC) Framework The UC framework [Can01] captures the security of the protocol $\mathsf{2PC}^{T,s_U}$ with an ideal functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$. The functionality defines an ideal protocol execution, where an environment $\mathcal{Z}$ provides inputs to and reads outputs from the two parties. And the parties simply forward their inputs to and receive outputs from $\mathcal{F}_{\mathsf{2PC}}$ as specified in Figure 1.

$\mathsf{2PC}^{T,s_U}$

Function Independent Offline Phase:

1. Alice and Bob run T instances of the sub-protocol $\mathsf{AuthGarb}^U$ in parallel, where $U : \{0,1\}^{s_U} \times \{0,1\}^{\ell_{\times A}} \times \{0,1\}^{\ell_{\times B}} \to \{0,1\}^{\ell_y}$ is a universal circuit accepting function descriptions of size bounded by s_U. For each $t \in [T]$:
   - Alice obtains input labels $\mathbf{K}_P^{(t)}, \mathbf{K}_A^{(t)}, \mathbf{K}_B^{(t)}$ masks $\mathbf{a}_I^{(t)}$, and decryption information $D^{(t)}$.
   - Bob obtains authenticated garblings $\overline{U}^{(t)}$ and masks $\mathbf{b}_I^{(t)}$.
2. Alice first encrypts the labels for public inputs into $\mathsf{ct}_P = (\mathsf{Sel.pp}_P, \mathsf{ct}_{P,1}, \{\mathsf{ct}_{P,2}^{(t)}\}, \{\mathsf{ct}_{P,i,b}^{(t)}\})$:
   - Sample $\mathbf{l}_1 \leftarrow \mathcal{M}^{s_U}$, and $\mathbf{l}_2^{(t)} \leftarrow \mathcal{M}^{s_U}$ for $t \in [T]$, and compute

$$\forall i \in [s_U], t \in [T] \quad \mathsf{ct}_{P,i,b}^{(t)} = H'(\mathbf{l}_1[i] \cdot b + \mathbf{l}_2^{(t)}[i]) \oplus \mathbf{K}_P^{(t)}[i,b],$$

   - Run $\mathsf{Sel.pp}_P \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U})$ and compute

$$(\mathsf{st}_{P,1}, \mathsf{Sel.ct}_{P,1}) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1), \quad (\mathsf{st}_{P,2}^{(t)}, \mathsf{Sel.ct}_{P,2}^{(t)}) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2^{(t)}) \quad \forall t \in [T].$$

   Alice then analogously encrypts the labels for private inputs into ct_I (with another instance of batch select), and the decryption information into $\mathsf{ct}_O^{(t)} = D^{(t)} \oplus H(\mathsf{seed}^{(t)})$ using random seeds. Alice sends $\mathsf{ct}_P, \mathsf{ct}_I, \{\mathsf{ct}_O^{(t)}\}$ to Bob.

t-th Funtion Dependent Offline Phase:

1. Alice sends a batch select decryption key $\mathsf{sk}_P^{(t)}$ computed as follows to Bob.

$$\mathsf{sk}_P^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{P,1}, \mathsf{st}_{P,2}^{(t)}, f_A^{(t)}).$$

2. Bob locally decrypts labels $\mathbf{k}_P^{(t)}$, where

$$\mathbf{l}^{(t)} \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_P^{(t)}, \mathsf{Sel.ct}_{P,1}, \mathsf{Sel.ct}_{P,2}^{(t)}, f_B^{(t)}), \quad \mathbf{k}_P^{(t)}[i] = \mathsf{ct}_{P,i,f_{B,i}}^{(t)} \oplus H'(\mathbf{l}^{(t)}[i]). \tag{24}$$

t-th Online Phase:

1. Bob sends $\overline{\mathbf{x}}_B^{(t)} = \mathbf{b}_I^{(t)} \oplus \mathbf{x}_B^{(t)}$ to Alice.
2. Alice sends $\overline{\mathbf{x}}_A^{(t)}, \mathsf{sk}_I^{(t)}, \mathsf{seed}^{(t)}$ to Bob, where

$$\mathsf{sk}_I^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{I,1}, \mathsf{st}_{I,2}^{(t)}, \overline{\mathbf{x}}_A^{(t)} \,\|\, \overline{\mathbf{x}}_B^{(t)}), \quad \overline{\mathbf{x}}_A^{(t)} = \mathbf{a}_I^{(t)} \oplus \mathbf{x}_A^{(t)},$$

3. Bob locally recover labels $\mathbf{k}_A^{(t)}, \mathbf{k}_B^{(t)}$ (analogously to Equation 24) and decryption information $D^{(t)} = \mathsf{ct}_O^{(t)} \oplus H(\mathsf{seed}^{(t)})$, and evaluates $\mathbf{y} = \mathsf{AuthEval}^U(\overline{U}^{(t)}, D^{(t)}, \mathbf{k}_P^{(t)}, \mathbf{k}_A^{(t)}, \mathbf{k}_B^{(t)}, C_B, \overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$.

Fig. 2: Our preprocessing T-session malicious 2PC protocol.

The ideal adversary/simulator Sim lets $\mathcal{F}_{\mathsf{2PC}}$ know of a corrupted party in the beginning, and then interacts with $\mathcal{F}_{\mathsf{2PC}}$ according to the corresponding adversarial interface. In additional

to explicitly specified adversarial interfaces, Sim receives all messages sent to and determins all outgoing messages from the corrupted party. The environment $\mathcal{Z}$ communicates with Sim freely throughout the protocol execution.

To prove the security of the protocol $\mathsf{2PC}$, we show that the ideal protocol specified above *emulates* a real protocol execution with an adversary $\mathcal{A}$ controlling a corrutped party, the honest party, and the environment $\mathcal{Z}$. We describe the real protocol execution, and the meaning of emulation below.

In the real protocol execution, the environment $\mathcal{Z}$ provides inputs to and reads outputs from the actual protocol participants. An adversary $\mathcal{A}$ decides a corrupted party in the beginning, and controls them throughout the protocol. The environment $\mathcal{Z}$ also communicates with $\mathcal{A}$ freely throughout the protocol execution.

We say that an ideal protocol execution with an ideal adversary Sim emulates a real protocol execution with an adversary $\mathcal{A}$, if no environment $\mathcal{Z}$ can tell whether it's interacting in the ideal or the real protocol. We say the protocol $\mathsf{2PC}^{T,s_U}$ UC-realizes the functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$ if for all efficient adversary $\mathcal{A}$, there exists an efficient ideal adversary Sim such that the ideal protocol with Sim emulates the real protocol with $\mathcal{A}$.

Formally, let $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$ and $\mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$ denotes the output of an environment after interacting in the ideal and the real protocol. We require that for all efficient $\mathcal{A}$, there exists an efficient Sim such that for all efficient $\mathcal{Z}$:

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}} \approx_c \mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}.$$

The UC framework allows for a modular presentation of protocols, thanks to the universal composition theorem. Consider an inner protocol π_{in} that UC-realizes an inner functionality $\mathcal{F}_{\mathsf{in}}$, and an outer protocol π_{out} that has access to copies of $\mathcal{F}_{\mathsf{in}}$, i.e., in a $\mathcal{F}_{\mathsf{in}}$-hybrid model, and UC-realizes another outer functionality $\mathcal{F}_{\mathsf{out}}$. The composition theorem ensures the composition of π_{out} and π_{in}, i.e., replacing each copy of $\mathcal{F}_{\mathsf{in}}$ with an instance of π_{i}, still UC-realizes $\mathcal{F}_{\mathsf{out}}$.

Proof of Theorem 5 We describe an ideal adversary Sim that externally interacts with the functionality $\mathcal{F}_{\mathsf{2PC}}$ and the environment $\mathcal{Z}$, while internally simulates a protocol execution with an instance of the adversary $\mathcal{A}$. When interacting with $\mathcal{Z}$, Sim simply forwards all communication between $\mathcal{A}$ and $\mathcal{Z}$. We consider separately the case when Alice or Bob is corrupted.

When Bob is Corrupted. Sim proceed as follows.

Instance Independent Offline Phase:

- Sim plays the role of $\mathcal{F}_{\mathsf{pre}}$ with Bob following its description (Figure 5). In the process, Sim samples its own global MAC key $\Delta_A \leftarrow \{0,1\}^\lambda$, input masks $\mathbf{a}_I^{(t)}$, and output masks $\{a^{(w,t)}\}_{w\in O}$ and tags $\{M_A^{(w,t)}\}_{w\in O}$. It also receives Bob's global MAC key $\Delta_B \in \{0,1\}^\kappa$ and input masks $\mathbf{b}_I^{(t)}$.
- Sim follows the description of $\mathsf{AuthGarb}$ (Figure 3) to compute and send garbled tables to Bob. In the process, Sim samples and stores input labels $\mathbf{K}_P^{(t)}, \mathbf{K}_A^{(t)}, \mathbf{K}_B^{(t)}$.
- Sim runs the (stateful) simulator $\mathsf{Sel.Sim}$ to simulate the batch select ciphertexts $\mathsf{ct}_P = (\mathsf{Sel.pp}_P, \mathsf{Sel.ct}_{P,1}, \{\mathsf{Sel.ct}_{P,2}^{(t)}\}, \{\mathsf{ct}_{P,i,b}^{(t)}\})$ for public inputs:

$$\mathsf{Sel.pp}_P \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}), \quad \mathsf{Sel.ct}_{P,1} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}_P),$$
$$\mathsf{Sel.ct}_{P,2}^{(t)} \leftarrow \mathsf{Sel.Sim}() \quad \text{for } t \in [T],$$

and then compute the ciphertexts $\{\mathsf{ct}_{i,b}^{(t)}\}$ honestly:

$$\mathbf{l}_{P,1} \leftarrow \mathcal{M}^{s_U}, \quad \mathbf{l}_{P,2}^{(t)} \leftarrow \mathcal{M}^{s_U} \quad \forall t \in [T]$$
$$\mathsf{ct}_{P,i,b}^{(t)} = H(\mathbf{l}_{P,1}[i] \cdot b + \mathbf{l}_{P,2}^{(t)}[i]) \oplus \mathbf{L}_P^{(t)}[i,b], \quad \forall i \in [s_U], t \in [T]$$

It analogously simulates the ciphertexts ct_I for private inputs, and then the ciphertext for decryption information at random $\mathsf{ct}_O^{(t)} \leftarrow \$$.

It sends honestly computed public parameters $\mathsf{Sel.pp}$, ciphertexts $\{\mathsf{ct}_{i,b}^{(t)}\}$ and the simulated ciphertexts $\mathsf{Sel.ct}_1$, $\{\mathsf{Sel.ct}_2^{(t)}\}$ to Bob.

t-th Function Dependent Offline Phase: In order to simulate the t-th batch select decryption key, Sim waits for the functionality $\mathcal{F}_{\mathsf{2PC}}$ to leak the target circuit $f_A^{(t)}$, and runs the (stateful) simulator $\mathsf{Sel.Sim}$:

$$\mathsf{sk}_P^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{l}_{P,1} \odot f_A^{(t)} + \mathbf{l}_{P,2}^{(t)}).$$

It sends the simulated decryption key $\mathsf{sk}_P^{(t)}$ to Bob.

t-th Online Phase:

- Sim receives masked inputs $\overline{\mathbf{x}}_B$ from Bob, and extracts the actual input $\mathbf{x}_B^{(t)} = \overline{\mathbf{x}}_B \oplus \mathbf{b}_I^{(t)}$. Sim then sends a message $(t, f_A^{(t)}, \mathbf{x}_B^{(t)})$ to the functionality $\mathcal{F}_{\mathsf{2PC}}$.
- Sim simulates the masked inputs from Alice as $\overline{\mathbf{x}}_A = \mathbf{a}_I^{(t)} \oplus \mathbf{0}$, and the batch select decryption key $\mathsf{sk}_I^{(t)}$ as

$$\mathsf{sk}_I^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{l}_{I,1} \odot \overline{\mathbf{x}}_A \,\|\, \overline{\mathbf{x}}_B + \mathbf{l}_{I,2}^{(t)}).$$

- In order to simulate the decryption information $D^{(t)}$, Sim waits for the functionality to reveal the evaluation result $\mathbf{y}^{(t)}$, and program $D^{(t)}$ so that evaluation on the simulated input labels correctly reveals $\mathbf{y}^{(t)}$. Sim first computes a difference vector

$$\delta = f_A(\mathbf{0}, \mathbf{x}_B^{(t)}) \oplus \mathbf{y}^{(t)},$$

and then adjusts Alice's bits $a^{(w,t)}$ and tags $M_A^{(w,t)}$ on output wires: (We abuse notations to write $\delta[w]$ to mean the difference bit in δ corresponding to the output wire $w \in O$.)

$$\forall w \in O, \text{ if } \delta[w] = 1, \quad a^{(w,t)} \leftarrow a^{(w,t)} \oplus 1, \quad M_A^{(w,t)} \leftarrow M_A^{(w,t)} \oplus \Delta_B.$$

Sim sets $D^{(t)} = \{a^{(w,t)}, M_A^{(w,t)}\}_{w \in O}$, samples a random $\mathsf{seed}^{(t)}$, and programs the random oracle $H(\mathsf{seed}^{(t)}) \leftarrow \mathsf{ct}_O^{(t)} \oplus D^{(t)}$. Finally, Sim sends $\overline{\mathbf{x}}_A^{(t)}$, $\mathsf{sk}_I^{(t)}$, and $\mathsf{seed}^{(t)}$ to Bob.

It remains to show the output of any environment $\mathcal{Z}$ in the a real protocol execution is indistinguishable from that in an ideal execution, i.e. $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}} \approx_c \mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$. We describe a series of hybrid experiments that transitions from $\mathsf{Hyb}_0 = \mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$ to $\mathsf{Hyb}_3 = \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$. We abuse the notation to also write Hyb_i as the output distribution of the environment.

Hyb_0: This is the real protocol execution between an honest Alice and a corrupted Bob in the $\mathcal{F}_{\mathsf{pre}}$-hybrid and random oracle model.

Hyb_1: Instead of honestly computing the batch select ciphertexts $\mathsf{Sel.ct}_{P,1}, \{\mathsf{Sel.ct}_{P,2}^{(t)}\}$ and decryption keys $\mathsf{Sel.sk}_P^{(t)}$ for public inputs, Alice runs the (stateful) batch select simulator as follows.

- In the instance independent offline phase, run

$$\mathsf{Sel.ct}_{P,1} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}_P), \quad \text{for } t \in [T], \mathsf{Sel.ct}_{P,2}^{(t)} \leftarrow \mathsf{Sel.Sim}().$$

- In the function dependent offline phase, run

$$\mathsf{sk}_P^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{I}_{P,1} \odot f_A^{(t)} + \mathbf{I}_{P,2}^{(t)}).$$

The adaptive T-times simulation security of Sel guarantees that the Hyb_1 is computationally indistinguishable from Hyb_0.

Hyb_2: Simulate the batch select ciphertexts and decryption keys for private inputs analogously to the previous hybrid:

$$\mathsf{Sel.ct}_{I,1} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}_I), \quad \text{for } t \in [T], \mathsf{Sel.ct}_{I,2}^{(t)} \leftarrow \mathsf{Sel.Sim}().$$
$$\mathsf{sk}_I^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{I}_{I,1} \odot \overline{\mathbf{x}}_A \,\|\, \overline{\mathbf{x}}_B + \mathbf{I}_{I,2}^{(t)}).$$

The adaptive T-times simulation security of Sel guarantees that the Hyb_2 is computationally indistinguishable from Hyb_1.

Hyb_3: In the instance independent offline phase, instead of computing $\mathsf{ct}_O^{(t)}$ honestly as $\mathsf{ct}_O^{(t)} = D^{(t)} \oplus H(\mathsf{seed}^{(t)})$, sample $\mathsf{ct}_O^{(t)}$ directly at random. Then in the online phase, program the random oracle phase as $H(\mathsf{seed}^{(t)}) \leftarrow \mathsf{ct}_O^{(t)} \oplus D^{(t)}$.

Since $\mathsf{seed}^{(t)}$ is sampled at random, the adversary has only negligable probability of querying $H(\mathsf{seed}^{(t)})$ before obtaining $\mathsf{seed}^{(t)}$ in the online phase. Therefore, Hyb_3 is statistically indistinguishable from Hyb_2.

Hyb_4: Alice additionally plays the role of $\mathcal{F}_{\mathsf{pre}}$ which allows her to learn Bob's global MAC key Δ_B and input masks $\mathbf{b}_I^{(t)}$ submitted to $\mathcal{F}_{\mathsf{pre}}$ during the instance independent offline phase.

Then during the t-th online phase, Alice receives a masked input $\overline{\mathbf{x}}_B^{(t)}$ from Bob, and extracts an input $\mathbf{x}_B^{(t)} = \overline{\mathbf{x}}_B^{(t)} \oplus \mathbf{b}_I^{(t)}$. Alice computes $\mathbf{y}^{(t)} = f_A^{(t)}(\mathbf{x}_A, \mathbf{x}_B)$, and program the decryption information $D^{(t)}$ such that the evaluation result of $\mathsf{AuthEval}$ equals $\mathbf{y}^{(t)}$.

Note that Hyb_4 is distributed identically to Hyb_3 because Alice is still computing her masked inputs honestly as $\overline{\mathbf{x}}_A^{(t)} = \mathbf{x}_A^{(t)} \oplus \mathbf{a}_I^{(t)}$, and the programmed decryption information $D^{(t)}$ is the same as the honestly computed.

Hyb_5: In the input phase, Alice simulates her masked inputs as $\overline{\mathbf{x}}_A^{(t)} = \mathbf{a}_I^{(t)} \oplus \mathbf{0}$, independent of her actual inputs $\mathbf{x}_A^{(t)}$, and then program the decryption information $D^{(t)}$ such that $\mathsf{AuthEval}$ still evaluates to the correct result $\mathbf{y}^{(t)} = f_A^{(t)}(\mathbf{x}_A, \mathbf{x}_B)$.

Note that this hybrid is distributed identically to $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$. The fact that Hyb_5 is statistically indistinguishable from Hyb_4 (in the random oracle model) follows from the same proof as Theorem 5.1 in [WRK17]. Hence we omit details here.

When Alice is Corrupted Sim proceed as follows.

Instance Independent Offline Phase:

- Sim plays the role of $\mathcal{F}_{\mathsf{pre}}$ with Alice following its description. In the process, Sim samples its own input masks $\mathbf{b}_I$, and receives Alice's input masks $\mathbf{a}_I^{(t)}$.
- Sim receives garbled tables and batch select ciphertexts from Alice and store them.

t-th Function Dependent Offline Phase: Sim receives a batch select decryption key $\mathsf{sk}_P^{(t)}$ from Alice, and use it to recover input labels $\mathbf{k}_P^{(t)}$ following the description of $\mathsf{2PC}$.

t-th Online Phase:

- Sim simulates its masked inputs as $\overline{\mathbf{x}}_B^{(t)} = \mathbf{b}_I \oplus \mathbf{0}$, and sends $\overline{\mathbf{x}}_B$ to Alice.
- Sim receives, besides a masked input $\overline{\mathbf{x}}_A$, a batch select decryption key $\mathsf{sk}_I^{(t)}$ and a $\mathsf{seed}^{(t)}$, and use them to recover input labels $\mathbf{k}_A^{(t)}, \mathbf{k}_B^{(t)}$ and decryption information $D^{(t)}$.
- Sim waits for the functionality $\mathcal{F}_{\mathsf{2PC}}$ to leak the target circuit $f_B^{(t)}$, and then runs $\mathsf{AuthEval}$ to check whether the evaluation prcedure aborts.
  - If $\mathsf{AuthEval}$ doesn't abort, then extract Alice's input as $\mathbf{x}_A^{(t)} = \overline{\mathbf{x}}_A^{(t)} \oplus \mathbf{a}_I^{(t)}$, and send a message $(t, f_B^{(t)}, \mathbf{x}_A^{(t)})$ to the functionality $\mathcal{F}_{\mathsf{2PC}}$.
  - If $\mathsf{AuthEval}$ aborts, then trigger an abort for the functionality $\mathcal{F}_{\mathsf{2PC}}$ by sending a message $(t, \emptyset, \emptyset)$.

The fact that $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$ is statistically indistinguishable from $\mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$ follows from the same proof as Theorem 5.1 in [WRK17]. Hence we omit details here.

$\mathsf{AuthGarb}^C$

The protocol assumes the $\mathcal{F}_{\mathsf{pre}}^{C,\lambda,\kappa}$ functionality defined in Figre 5.
Let W be the set of indices of all wires in C, and I_P, I_A, I_B, O be those of public inputs, Alice and Bob's inputs, and output wires. Let $W_\wedge$ be the output wires of all AND gates.

1. Alice and Bob invoke $\mathcal{F}_{\mathsf{pre}}^{(C,\lambda,\kappa)}$.
   - Alice samples a global MAC key $\Delta_A \leftarrow \{0,1\}^\lambda$, bits $a^{(w)} \leftarrow \{0,1\}$, tags $M_A^{(w)} \leftarrow \{0,1\}^\kappa$ for $w \in W$, and sets $a^{(w)} = 0$ for $w \in I_B \cup I_P$.
   - Bob analogously samples $\Delta_B, \{b^{(w)}\}, \{M_A^{(w)}\}$, and sets $b^{(w)} = 0$ for $w \in I_A \cup I_P$.

   They send $(\Delta_A, \{a^{(w)}, M_A^{(w)}\})$ and $(\Delta_B, \{b^{(w)}, M_B^{(w)}\})$ to $\mathcal{F}_{\mathsf{pre}}$ and receive back $(\{K_A^{(w)}\}, \{\widehat{a}^{(w)}, \widehat{M}_A^{(w)}, \widehat{K}_A^{(w)}\})$ and $(\{K_B^{(w)}\}, \{\widehat{b}^{(w)}, \widehat{M}_B^{(w)}, \widehat{K}_B^{(w)}\})$.
2. Alice and Bob locally computes intermediate bits, tags, and MAC keys for each AND gate $(i, j, k, \wedge)$ as follows:
   - Alice computes

$$\begin{aligned}
\overline{a}_0^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)}, & \overline{a}_1^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)} \oplus a^{(i)},\\
\overline{a}_2^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)} \oplus a^{(j)}, & \overline{a}_3^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)} \oplus a^{(i)} \oplus a^{(j)} \oplus 1,\\
\overline{M}_{A,0}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)}, & \overline{M}_{A,1}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)} \oplus M_A^{(i)}\\
\overline{M}_{A,2}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)} \oplus M_A^{(j)}, & \overline{M}_{A,3}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)} \oplus M_A^{(i)} \oplus M_A^{(j)},\\
\overline{K}_{A,0}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)}, & \overline{K}_{A,1}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)} \oplus K_A^{(i)}\\
\overline{K}_{A,2}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)} \oplus K_A^{(j)}, & \overline{K}_{A,3}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)} \oplus K_A^{(i)} \oplus K_A^{(j)} \oplus \Delta_A.
\end{aligned}$$

   - Bob analoguously computes $\overline{b}_d^{(k)}, \overline{M}_{B,d}^{(k)}, \overline{K}_{B,d}^{(k)}$ for $d \in [3]$.
3. Alice compute garbled tables as follows and sends them to Bob.
   - For each wire $w \in W$, sample a random key $L_0^{(w)} \leftarrow \{0,1\}^\lambda$, and set $L_1^{(w)} = L_0^{(w)} \oplus \Delta_A$.
   - For each AND gate $(i, j, k, \wedge)$, compute a garbled table $\mathsf{tb}^{(k)} = (\mathsf{ct}_0^{(k)}, \mathsf{ct}_1^{(k)}, \mathsf{ct}_2^{(k)}, \mathsf{ct}_3^{(k)})$ where for $d \in [3]$

$$\mathsf{ct}_d^{(k)} = H(L_{d_0}^{(i)} \| L_{d_1}^{(j)} \| d) \oplus \left(\overline{a}_d^{(k)} \| \overline{M}_{A,d}^{(k)} \| L_0^{(k)} \oplus \overline{a}_d^{(k)} \cdot \Delta_A \oplus \overline{K}_{A,d}^{(k)}\right)$$

4. Alice outputs
   - the input keys $L_0^{(w)}, L_1^{(w)}$ for $w \in I_P \cup I_A \cup I_B$, organized into $\mathbf{K}_P, \mathbf{K}_A, \mathbf{K}_B$;
   - the bits $a^{(w)}$ for $w \in I_A$ organized as a vector $\mathbf{a}_I$;
   - decryption information $D = \{a^{(w)}, M_A^{(w)}\}_{w \in O}$.

   Bob outputs
   - the authenticated garbling $\widehat{C} = (\Delta_B, \{\mathsf{tb}^{(k)}\}, \{\overline{b}_d^{(k)}, \overline{M}_{B,d}^{(k)}, \overline{K}_{B,d}^{(k)}\})$;
   - the bits $b^{(w)}$ for $w \in I_B$ organized as a vector $\mathbf{b}_I$.

Fig. 3: Sub-protocol $\mathsf{AuthGarb}$, adapted from [WRK17, DILO22].

$\mathsf{AuthEval}^C(\overline{C}, D, \mathbf{k}_P, \mathbf{k}_A, \mathbf{k}_B, \overline{\mathbf{x}}_P, \overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$

Let W be the set of indices of all wires in C, and I_P, I_A, I_B, O be those of public inputs, Alice and Bob's inputs, and output wires. Let $W_\wedge$ be the output wires of all AND gates.

1. Parse the inputs as:

$$\overline{C} = (\Delta_B, \{\mathsf{tb}^{(k)}\}_{k\in W_\wedge}, \{\overline{b}_d^{(k)}, \overline{M}_{B,d}^{(k)}, \overline{K}_{B,d}^{(k)}\}_{k\in W_\wedge, d\in[3]}),$$
$$\mathsf{tb}^{(k)} = \{\mathsf{ct}_d^{(k)}\}_{d\in[3]}, \quad D = \{a^{(w)}, M_A^{(w)}\}_{w\in O}.$$

2. In topological order, for every gate (i, j, k, Type) the algorithm holds $L^{(i)}, L^{(j)}$ and bits $\overline{x}^{(i)}, \overline{x}^{(j)}$, and proceed as follows:
   - If Type is $\oplus$, then compute
$$L^{(k)} = L^{(i)} \oplus L^{(j)}, \quad \overline{x}^{(k)} = \overline{x}^{(i)} \oplus \overline{x}^{(j)}.$$
   - If Type is $\wedge$, first decrypt the ciphertext $\mathsf{ct}_d^{(k)}$ where $d = 2 \cdot \overline{x}^{(i)} + \overline{x}^{(j)}$:
$$(\overline{a}_d^{(k)} \| \overline{M}_d^{(k)} \| L_A^{(k)}) = \mathsf{ct}_d^{(k)} \oplus H(L^{(i)} \| L^{(j)} \| d),$$
     Next verify that $\overline{M}_d^{(k)} = \overline{a}_d^{(k)} \cdot \Delta_B \oplus \overline{K}_{B,d}^{(k)}$, and then compute
$$\overline{x}^{(k)} = \overline{a}_d^{(k)} \oplus \overline{b}_d^{(k)}, \quad L^{(k)} = L_A^{(k)} \oplus \overline{M}_{B,d}^{(k)}.$$

3. In the end, the algorithm holds $\overline{x}^{(w)}$ for every $w \in O$, and proceeds as follows.
   - For every $w \in O$, verify that $M_A^{(w)} = a^{(w)} \cdot \Delta_B \oplus K_B^{(w)}$, and compute $x^{(w)} = \overline{x}^{(w)} \oplus a^{(w)} \oplus b^{(w)}$.
   - Output the values on the output wires as $\mathbf{y}$.

Fig. 4: Algorithm $\mathsf{AuthEval}$, adapted from [WRK17, DILO22].

Functionality $\mathcal{F}_{\mathsf{pre}}^{(C,\lambda,\kappa)}$

Let W be the set of indices of all wires in C, $I, O \subset W$ be the indices of input and outptu wires, and $W_\wedge$, output wires of AND gates. Further divide I input indices for public inputs I_P, Alice's inputs I_A and Bob's inputs I_B.

- On receiving $(\Delta_A \in \{0,1\}^\lambda, \{a^{(w)} \in \{0,1\}, M_A^{(w)} \in \{0,1\}^\kappa\}_{w \in W})$ from Alice, or $(\Delta_B \in \{0,1\}^\kappa, \{b^{(w)} \in \{0,1\}, M_B^{(w)} \in \{0,1\}^\lambda\}_{w \in W})$ from Bob, store them.
- Once received messages from both Alice and Bob:
  1. For every input wire $w \in I_A$, adjust Bob's bit $b^{(w)} = 0$. For every input wire $w \in I_B$, adjust Alice's bit $a^{(w)} = 0$.
  2. For every XOR gate $(i, j, k, \oplus)$, adjust the bits and tags on the output wire k:

$$\begin{aligned} a^{(k)} &= a^{(i)} \oplus a^{(j)}, & M_A^{(k)} &= M_A^{(i)} \oplus M_A^{(j)}, \\ b^{(k)} &= b^{(i)} \oplus b^{(j)}, & M_B^{(k)} &= M_B^{(i)} \oplus M_B^{(j)}. \end{aligned}$$

  3. For every AND gate $(i, j, k, \wedge)$, sample random additive shares $\widehat{a}^{(k)}, \widehat{b}^{(k)} \in \{0,1\}$ such that
$$\widehat{a}^{(k)} \oplus \widehat{a}^{(k)} = (a^{(i)} \oplus b^{(i)}) \cdot (a^{(j)} \oplus b^{(j)}),$$
  and random tags $\widehat{M}_A^{(k)} \leftarrow \{0,1\}^\kappa, \widehat{M}_B^{(k)} \leftarrow \{0,1\}^\lambda$.
  4. For every tag, compute the corresponding mac key:

$$\begin{aligned} \forall w \in W, \quad & K_B^{(w)} = M_A^{(w)} \oplus a^{(w)} \cdot \Delta_B \quad K_A^{(w)} = M_B^{(w)} \oplus b^{(w)} \cdot \Delta_A \\ \forall k \in W_\wedge, \quad & \widehat{K}_B^{(k)} = \widehat{M}_A^{(k)} \oplus \widehat{a}^{(k)} \cdot \Delta_B \quad \widehat{K}_A^{(k)} = \widehat{M}_B^{(k)} \oplus \widehat{b}^{(k)} \cdot \Delta_A. \end{aligned}$$

  Output $(\{K_A^w\}, \{\widehat{a}^{(w)}, \widehat{M}_A^{(w)}, \widehat{K}_A^w\})$ and $(\{K_B^w\}, \{\widehat{b}^{(k)}, \widehat{M}_B^{(k)}, \widehat{K}_B^k\})$ to Alice and Bob respectively.

Fig. 5: The preprocessing functionality adapted from [WRK17, DILO22].