Succinct Garbled Circuits with Low-Depth Garbling Algorithms

Hanjun Li
CMU
hanjunl@andrew.cmu.edu

Huijia Lin
University of Washington
rachel@cs.washington.edu

George Lu
UT Austin
gclu@cs.utexas.edu

Abstract

We study the problem of constructing Boolean garbling schemes that are both **succinct**—with garbled circuit size significantly smaller than the original circuit—and have **low-depth** garbling algorithms, where the garbling process runs in parallel time logarithmic in the circuit size. Prior schemes achieve one but not the other, unless relying on indistinguishability obfuscation (iO), which is prohibitively inefficient, relies on a combination of multiple assumptions, and achieves only polynomial garbling depth $\mathsf{poly}(\lambda, \log |C|)$.

We resolve this tension by presenting the first garbling schemes that are both succinct and admit garbling algorithms in NC^1, based only on standard group and lattice assumptions. Our main results include:

- **One-bit-per-gate garbling** with logarithmic garbling depth based on DDH or RLWE and the existence of a local PRG.
- Succinct **Privacy-free garbling** of size linear in the circuit depth D (and sublinear in the circuit size $|C|$), based on DDH or RLWE.
- **Reusable, fully succinct garbling** with logarithmic garbling depth, based on decomposable LWE.

The DDH-based one-bit-per-gate scheme has tunably small inverse polynomial correctness and privacy errors, which can be made negligible at the cost of increasing garbling depth to $\mathsf{poly}(\lambda)$.

As further extension, we also obtain the first attribute-based encryption schemes with *succinct keys* and *low-depth* key generation.

At a conceptual level, our constructions are derived from a **unified framework** that subsumes all prior approaches to succinct garbling. It identifies the common source of high-depth garbling, and provides a general methodology for reducing garbling depth without sacrificing succinctness, applicable across different techniques and assumptions.

Contents

1 Introduction **3**
- 1.1 Related and Concurrent Works . . . 6

2 Technical Overview **7**

3 Definitions **10**
- 3.1 Garbled Circuits with Algebraic Output Labels . . . 13

4 A General Template for Low-Depth Succinct Garbling **16**
- 4.1 Garbling Depth . . . 17

5 Non-Reusable Succinct Privacy-Free Garbling **18**
- 5.1 Preliminaries . . . 18
- 5.2 Instantiation Under Groups . . . 20
- 5.3 Instantiation Under Lattices . . . 31

6 Non-Reuseable Partially Succinct Garbling **38**
- 6.1 Instantiation Under Groups . . . 38
- 6.2 Instantiation Under Lattices . . . 42

7 Reusable Succinct Privacy-Free Garbling **44**
- 7.1 Preliminaries . . . 44
- 7.2 Construction . . . 47
- 7.3 Garbler Depth Complexity . . . 49
  - 7.3.1 Log-Depth Preimage Sampling . . . 49
  - 7.3.2 Log-Depth Key-Homomorphic RAM Encoding . . . 50
- 7.4 Succinctness and Decomposability . . . 50
- 7.5 Correctness . . . 50
- 7.6 Security . . . 51

8 Reusable Fully Succinct Garbling **54**
- 8.1 Preliminaries . . . 55
- 8.2 Construction . . . 56
- 8.3 Garbler Depth Complexity . . . 57
- 8.4 Succinctness and Decomposability . . . 57
- 8.5 Correctness . . . 57
- 8.6 Security . . . 57

9 Attribute-Based Encryption **59**
- 9.1 Definitions . . . 60
- 9.2 Construction . . . 60
- 9.3 Key Succinctness and Generation Depth . . . 62
- 9.4 Correctness . . . 62
- 9.5 Security . . . 63

10 Acknowledgments **67**

1 Introduction

A Boolean garbling scheme transforms a Boolean circuit C and an input x into garbled versions, $(\hat{C}, \mathsf{gk}) \leftarrow \mathsf{GarbC}(1^\lambda, C)$ and $\hat{x} \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x)$, from which the output can be recovered $\mathsf{Eval}(C, \hat{C}, \hat{x}) = y$ without revealing any other information about the input. Garbling schemes are a foundational primitive in cryptography with wide-ranging applications, including multiparty computation [Yao86, BMR90, IPS08, GS17, GS18, BL18], parallel cryptography [AIK04, AIK06], and verifiable computation [GGP10, AIK10] etc; see [App17] for a survey.

Given their central role, a huge amount of work has focused on improving the efficiency of garbled circuits. Two aspects have received particular attention: succinctness, namely minimizing the size of the garbled circuit and garbled inputs; and low-depth garbling, namely reducing the depth, and thus the parallel-time, of the garbling algorithms. Surprisingly, almost all known constructions achieve either succinctness *or* low-depth garbling but not both, except for schemes relying on indistinguishability obfuscation (iO). In this work, we resolve this tension by presenting an array of *succinct* and *low-depth* garbling schemes, based on standard group and lattice assumptions, that are significantly simpler and more efficient than iO-based approaches.

Succinctness. Yao's original construction [Yao82] and subsequent optimizations [BMR90, NPS99, KS08, PSSW09, KMR14, GLNP15, ZRE15, RR21] remain the most computationally efficient general-purpose garbling schemes, owing to their reliance on fast symmetric-key cryptography. They also achieve extremely low garbling depth. The main limitation is their communication complexity: the current state of the art, due to Rosulek and Roy [RR21], devotes $1.5\lambda + 5$ bits to garbling an AND gate, while XOR and NOT gates are free under the random oracle model.

This $\Omega(\lambda)$-bits-per-gate barrier motivated the study of *succinct* garbled circuits, where the bit-length of the garbled circuit $|\hat{C}|$ is asymptotically smaller than the description size of the original circuit C. Since a circuit C with $|C|$ gates requires at least $|C| \log |C|$ bits to describe, succinct garbling demands $|\hat{C}| < |C| \log |C|$[1]. Two main approaches have emerged:

- *Fully succinct garbling schemes.* These achieve garbled circuit size independent of $|C|$. Constructions either rely on iO [KLW15, BCG+18], or on combinations of homomorphic encryption (HE) and attribute-based encryption (ABE) [GKP+13, BGG+14, HLL23]. In addition to optimal asymptotic size, these schemes are *reusable* [GKP+13], meaning the same garbled circuit can be used to evaluate arbitrarily many input encodings. However, their concrete efficiency is poor. iO-based constructions are prohibitively expensive, while FHE-plus-ABE approaches are more efficient in a relative sense, they are still far from practical.

- *One-bit-per-gate garbling schemes.* A more recent line of work [LWYY25, ILL25b, MORS25b] constructs garbled circuits of size $|\hat{C}| = |C| + o(|C|)$, achieving essentially one bit per gate. These schemes are significantly more efficient than fully succinct ones and have already demonstrated promising concrete performance [ILL25b, LWYY25]. This improvement stems from their reliance on comparatively lightweight techniques, such as homomorphic secret sharing (HSS) [BGI16] and FHE without bootstrapping [GSW13], rather than heavy tools like iO or the non-black-box combination of FHE and ABE. A further advantage is flexibility in assumptions: one-bit-per-gate constructions can be based on a diverse set of algebraic assumptions, including RLWE, NTRU, DDH over prime-order or Paillier groups, and DCR, in contrast to the narrow reliance of fully succinct approaches on iO or lattice-based primitives.

Low-depth garbling. A major drawback of existing succinct garbling schemes in either of these categories is that their garbling algorithms have high depth, growing polynomially with the circuit depth D (namely $\mathsf{poly}(\lambda, D)$). The only exception is constructions based on iO, which achieve garbling depth $\mathsf{poly}(\lambda)$ but at prohibitive concrete cost.

In contrast, a low-depth garbling scheme is one where the garbling depth depends only logarithmically on the circuit size, that is, $\mathsf{poly}(\lambda, \log |C|)$. Such schemes enable highly parallelizable garbling, which are crucial in applications where garbling depth directly translates into round complexity and latency. Yao's garbled circuits exemplify this property: all gates can be garbled in parallel, yielding depth $\mathsf{poly}(\lambda)$ under one-way functions, $O(\log \lambda)$ assuming PRGs in NC^1, and even NC^0 when combined with an NC^0-garbling scheme for NC^1 circuits [IK02, AIK04].

[1] Note that C and $\hat{C}$ are syntactically different objects. C is a circuit, while $\hat{C}$ is a binary string. Adopting standard notation, $|C|$ denotes the number of *gates* in the original circuit C, while $|\hat{C}|$ denotes the bit-length of the garbled circuit.

This contrast raises the central questions of our work:

Can we construct Boolean garbling schemes that are simultaneously succinct *and* low-depth*, without relying on iO?*
Can succinct garbling be in NC^1*, which is unknown even assuming iO?*

Our Results. We give a comprehensive and affirmative answer to the above question by enhancing existing non-iO-based succinct garbling schemes to achieve low – indeed NC^1 – garbling depth while preserving succinctness. Concretely: 1) by enhancing the schemes of [ILL25b], we obtain one-bit-per-gate garbled circuits with NC^1 *garbling*, based on DDH or RLWE, and the existence of local Pseudorandom Generators (PRG) [Gol00]; and 2) by enhancing the "leveled" version of the scheme in [GKP+13, BGG+14], we obtain a *fully succinct, reusable* garbling scheme with NC^1 garbling, based on decomposable LWE [AMR25]. To the best of our knowledge, our succinct garbling schemes are the first to achieve NC^1 garbling depth; this level of parallel efficiency was not attained even by prior iO-based schemes, which have $\mathsf{poly}(\lambda)$ depth.

A key intermediate step towards these constructions is the design of new *privacy-free* garbling schemes that retain various degrees of succinctness while enjoying NC^1 *garbling*, based on the same assumptions. These schemes are of independent interest. Unlike standard garbling, privacy-free garbling [JKO13, FNO15] requires only computational integrity, not privacy: the input x is public, the evaluation algorithm takes it as input explicitly, and the scheme guarantees *authenticity*, namely that it is computationally hard to produce a valid label for an incorrect output (with respect to an authentication key). Privacy-free garbling and the closely related notion of partial garbling [IW14] have found many applications, including efficient zero-knowledge arguments for non-algebraic statements [JKO13, FNO15], CDS and PSM protocols, constrained PRFs [ILL25a], and blockchain applications [Lin25]. Yet in this setting as well, no prior construction achieved both succinctness and low-depth: the only known succinct privacy-free garbling [ILL25a] had depth proportional to the circuit depth. Our work fills this gap.

In essence, our constructions are summarized in the following informal theorems.

Theorem 1.1 (One-Bit-Per-Gate, NC^1-Garbling-Depth, Garbled Circuits, Informal)**.** *Assume DDH over prime-order subgroup of* $\mathbb{Z}_p^*$*, or DDH over Paillier groups, or RLWE, and the existence of constant-local PRGs. The constructions in Section 6 are garbling schemes for Boolean circuits with the following properties:*

- One-bit-per-gate: *The size of the garbled circuit is* $|\hat{C}| = |C| + \mathsf{poly}(\lambda) \cdot (D + o(|C|))$[2]*, where* $|C|$ *is the number of gates and* D *is the depth of the circuit.*
- NC^1 garbling: *The depth of the garbling algorithm is* $O(\log \lambda + \log |C|)$.

The instantiation based on DDH over subgroup of $\mathbb{Z}_p^*$ *has a* $1/poly(\lambda)$ *correctness and privacy errors. The errors can be made negligible, at the price of increasing the garbling depth to* $\mathsf{poly}(\lambda)$*. Moreover, assuming the same assumptions without local PRGs, the constructions in Section 5 yield* privacy-free *garbling schemes for Boolean circuits, with garbled circuit size* $\mathsf{poly}(\lambda) \cdot (D + o(|C|))$ *and garbling algorithm depth* $O(\log \lambda + \log |C|)$.

We note that achieving NC^1 garbling requires the ability to evaluate iterated basic operations of the underlying algebraic structure, such as repeated group operations or ring multiplications, in NC^1. For this reason, we assume DDH over subgroups of $\mathbb{Z}_p^*$. When instantiated under DDH over general groups, our schemes are still low-depth, but with depth complexity $\mathsf{poly}(\lambda)$ rather than NC^1.

Theorem 1.2 (Depth-Succinct, NC^1-Garbling-Depth, Reusable Garbled Circuits, Informal)**.** *Assume decomposable LWE. The construction in Section 8 is a* reusable *garbling scheme for Boolean circuits with the following properties:*

- Depth-succinctness: *The size of the garbled circuit is* $\mathsf{poly}(\lambda, D)$*, where* D *is the circuit depth.*
- NC^1 garbling: *The depth of the garbling algorithm is* $O(\log \lambda + \log |C|)$.

[2]The exact sublinear factors in $|C|$ varies per construction. Refer to the formal statements in Section 6 for the exact parameters.

The decomposable LWE assumption was recently introduced in [AMR25]; it can be viewed as a circular-security assumption for a natural lattice-based public-key encryption scheme and is implied by the hardness of the succinct LWE problem of [Wee24]. See Assumption 7.5 for the formal definition and [AMR25] for further discussion.

Theorems 1.1 and 1.2 improve upon prior one-bit-per-gate garbling [LWYY25, ILL25b, MORS25b] and succinct reusable garbling [GKP+13, BGG+14, QWW18] respectively by reducing the garbling depth from $\mathsf{poly}(\lambda, D)$ to $\mathsf{poly}(\log \lambda, \log |C|)$ while maintaining the same asymptotic garbling size and underlying assumptions (albeit restricted to integer groups for DDH).

Extending Theorem 1.2: ABE with Low-Depth and Succinct Key Generation. We further extend the above theorem to obtain the first attribute-based encryption (ABE) with succinct keys and low-depth key generation Section 9.

Reusable garbled circuits and attribute-based encryption (ABE) are closely connected. Existing reusable garbled circuits [GKP+13, BGG+14, QWW18] are typically constructed by combining homomorphic encryption (HE) and ABE, in such a way that the depth of the garbling algorithm mirrors the depth of the ABE key generation, and the garbled circuit size grows with the size of the ABE secret key. Since all prior ABE schemes with succinct secret keys [BGG+14, HLL23] had high-depth key generation, the corresponding reusable garbled circuits also had high-depth garbling. We show that the same techniques we use to reduce garbling depth also apply to ABE, yielding the first ABE scheme with both succinct keys and low-depth key generation.

Theorem 1.3 (ABE with Low-Depth and Succinct Key Generation, Informal)**.** *Assume decomposable LWE. The construction in Section 9 is an ABE scheme for Boolean circuits with the following properties:*

- Depth-succinct keys: *For a depth-D circuit C, the secret key has size* $\mathsf{poly}(\lambda, D)$.
- NC^1 key generation: *The depth of the key generation algorithm is* $O(\log \lambda + \log |C|)$.

Unified Framework. Our constructions are obtained through a unified framework (described in Section 4) that provides a general method for reducing garbling depth without sacrificing succinctness. This framework extends and abstracts the approach of [ILL25b], which combined homomorphic secret sharing (HSS) with algebraic HMAC (aHMAC) [ILL25a] to realize one-bit-per-gate garbled circuits. Unlike that specialized setting, our framework operates at a higher level of abstraction and encompasses both one-bit-per-gate garbling and lattice-based reusable garbling [GKP+13, BGG+14], despite their very different underlying tools (HSS and aHMAC in the former, HE and ABE in the latter).

The framework highlights a common source of high garbling depth in all prior succinct garbling schemes. Conceptually, it views garbling as a two-party protocol between the garbler and the evaluator, who jointly maintain secret shares of every wire value. Evaluation proceeds via local homomorphic computations on each party's shares (corresponding to the garbling and evaluation algorithms), together with one-way communication from the garbler to the evaluator (the garbled circuit itself). In this perspective, the "secret sauce" behind succinctness is the use of homomorphic techniques–HSS, aHMAC, HE, ABE–that reduce communication and thereby garbled circuit size.

The conflict with low-depth garbling arises because these homomorphic techniques typically require *symmetric* (but not identical) local computation by both the garbler and the evaluator, which results in high depth at both ends. Our key idea is to break this symmetry by introducing *periodic share refreshing*. Specifically, the garbler's share of an intermediate value is re-sampled from a fixed distribution, while the evaluator's share is updated accordingly. As a result, the garbler's computation becomes shallow, given all refreshed shares.

A challenge is that naively refreshing all intermediate shares incurs communication linear in $|C|$, destroying succinctness. To overcome this, we compress shares of intermediate values into compact shares of a *hash* and refresh only the latter, which incurs small communication and preserves succinctness. To implement this, we adapt the recent algebraic compression techniques of [AMR25, Wee24], originally developed for succinct garbled RAM and ABE with succinct ciphertexts, to the garbling setting, and provide new group-based instantiation.

Open Questions The garbling size of our schemes in Theorem 1.1 and Theorem 1.2 all exceed $D \cdot \mathsf{poly}(\lambda)$. Under stronger circular-security variants of the assumptions, prior works can reduce the garbled circuit size further, down to $|\hat{C}| = |C| + \mathsf{poly}(\lambda)$ for one-bit-per-gate garbling [LWYY25, ILL25b, MORS25b] and $\mathsf{poly}(\lambda)$ for reusable

garbling [HLL23]. Our schemes, however, do not currently admit analogous variants, and we leave closing this gap as an interesting open question.

Another natural question is whether the depth of our schemes can be further reduced to NC^0 using the standard composition technique of [IK02, AIK04]. This technique combines an NC^1 garbling algorithm for general circuits with an NC^0 garbling algorithm for NC^1 circuits. While this approach would indeed reduce the garbling depth to NC^0, it comes at the cost of losing succinctness, since existing NC^0 schemes are not succinct. Whether succinct garbling can be constructed in NC^0 is another extremely interesting question.

1.1 Related and Concurrent Works

This work focuses on garbling schemes for Boolean circuits. Garbling schemes for other computational models have also been studied, and in some cases achieve notions of succinctness and low-depth garbling simultaneously. We describe and compare with these below.

Comparison with Succinct-Time Garbled RAM [AMR25]. The work of [AMR25] (AMR) targets minimizing the *time* to garble RAM programs, whereas our work aims to reduce the *depth* (parallel time) for garbling Boolean circuits. Based on circular LWE and decomposable LWE, AMR shows how to garble a RAM program P in time $\mathsf{poly}(\lambda, |P|, \log T)$, logarithmic in the time complexity T of P. By contrast, garbling a circuit necessarily takes time linear in its size $|C|$, simply to read its description. Instead, we focus on reducing the parallel-time: our garbling algorithms run in logarithmic parallel time (NC^1), while the AMR garbling algorithm has depth $\mathsf{poly}(\lambda)$.

In addition, our constructions provide advantages beyond parallel-time. Our non-reusable schemes (Theorem 1.1) can be based on group assumptions as well as lattices, whereas AMR relies solely on lattice assumptions. Our reusable schemes (Theorem 1.2) achieve reusability, while the AMR construction does not. Despite these differences, our work is technically connected with [AMR25], as mentioned above and elaborated in later sections.

Comparison with Succinct Bounded-Integer Garbling [BLLL23]. The work of [BLLL23] presents a garbling scheme for arithmetic circuits (with addition and multiplication gates) over integers from a bounded range $[2^\ell]$, where ℓ is the bit length of the values. Their scheme achieves communication of $(O(\ell) + \mathsf{poly}(\lambda))$ bits per gate using linearly homomorphic encryption based on DCR. As in Yao's garbled circuits, their garbling algorithm samples fresh "keys" for each gate, so all gates can be garbled in parallel, yielding NC^1 garbling depth.

Follow-up work [MORS24] improved the communication to $\ell + \mathsf{poly}(\lambda)$ bits per gate using HSS techniques, but at the cost of losing low-depth garbling. However, these schemes do not yield succinct garbling for Boolean circuits: when $\ell = 1$, the communication remains $\mathsf{poly}(\lambda)$ bits per gate.

Comparison with Attribute-Based LFE from DCR [MORS25a]. The concurrent work of [MORS25a] presents an attribute-based laconic function evaluation (AB-LFE) scheme from DCR. Our works share some technical similarities - both works present new group-based instantiations of linear-hashing reminiscent of the lattice-based constructions of [Wee24, AMR25] and leverage the homomorphic properties of such a hash to achieve some notion of succinctness. However, beyond this shared technical core, our techniques and results building upon this hash largely diverge. [MORS25a] focuses on achieving a notion of sender time-succinctness in a two message computation setting corresponding to (AB-)LFE, while we focus on retaining size-succinctness in the setting of garbled circuits while reducing the depth complexity of the garbler.

We note that the constructions of neither paper do not directly imply the other - the AB-LFE scheme of [MORS25a] is not directly low-depth and our group-based garbling scheme does not achieve any notion of laconicity. However, due to the usage of essentially equivalent underlying hash compression, it seems likely the techniques can be applied in a fairly direct manner when relevant (e.g. modifying the [MORS25a] AB-LFE to additionally be low-depth using ideas presented in this work).

2 Technical Overview

The starting point of our ideas will be the high level view of succinct garbling in [MORS24, MORS25b, ILL25b], where a garbling scheme is constructed as a 2-party protocol where the garbler and evaluator jointly compute 'shares' of each wire w_i in the circuit $[\![w_i]\!]$. This view captures both privacy-free garbling (where the input is not hidden from the evaluator, who only seeks to produce an authenticated 'label' of the output) as well as the traditional notion of private garbling. In these protocol, only the garbler sends messages, which represents the garbled circuit. This roughly breaks down a garbling protocol into the following steps:

- $\mathsf{Enc}(x) \to [\![x]\!]$: Given an input x, the garbler generates an encoding $[\![x]\!]_G, [\![x]\!]_E$ of the input, and $[\![x]\!]_E$ is sent to the evaluator as the garbled input.
- $\mathsf{HGate}([\![w_1]\!], [\![w_2]\!], \mathsf{op}) \to [\![\mathsf{op}(w_1, w_2)]\!]$: With encodings of input wires, the garbler and evaluator can jointly generate encoding of the output of a gate. Any communication this may require on the garbler's part is part of the garbled circuit.
- $\mathsf{Dec}([\![C(x)]\!]) \to \mathsf{dk}$: Finally, given an encoding of the output wire, the garbler sends some decoding information which allows the evaluator to recover the output of the garbled circuit.

Zooming in a little further, in these prior schemes[ILL25b, MORS25b], the garbler's $[\![w_i]\!]_G = k_i$ and evaluator's $[\![w_i]\!]_E = \sigma_i = k_i + s \cdot w_i$ shares form an additive secret sharing of $s \cdot w_i$, for some garbling secret s. To advance the computation with some gate $\mathsf{op}(w_1, w_2)$, the garbler publishes some cryptographic material which enables the garbler and evaluator to run HGate to obtain additive shares of $s' \cdot \mathsf{op}(w_1, w_2)$ either completely *noninteractively* (in the case of privacy-free garbling) or with only a single bit of communication (in the setting of private garbling).[3] Importantly, this cryptographic material is reusable, and given shares of every depth j wire, enables the garbler and evaluator to obtain shares of *every* depth $j + 1$ wire, enabling succinctness. Moreover, this HGate protocol works on *any* additive sharing of $s \cdot w$.

Rekeying. This latter observation yields the following simple idea - the garbler 'fixes' their shares k_i of the additive encoding of all the circuit wires before performing any homomorphic evaluation. Since the garbler already has their 'shares' of every wire in the circuit, they can in parallel run HGate for every wire of C, albeit obtaining a 'different' additive share of the output wire k'_i. To ensure the evaluator can proceed with their computation, the garbler simply sends them $\mathsf{rk} = (k_i - k'_i)$, which allows the evaluator to realign or 'rekey' their additive share to be consistent with the garbler's independently sampled shares.

We formalize and abstract this idea through defining a 'rekeying' protocol, which takes the following form:

- $\mathsf{RekeySamp}(1^\lambda) \to k$: The garbler can sample a fresh share k
- $\mathsf{RekeyGen}(k'_i, k_i) \to \mathsf{rk}$: Given a share k'_i (which will be the output of HGate) and a 'target share' k_i (which will be independently sampled), the garbler computes a rekeying key.
- $\mathsf{RekeyEval}(\sigma'_i, \mathsf{rk}) \to \sigma_i$: When delivered to the evaluator, the rekey rk can be used in conjunction with a share σ'_i to produce a share σ_i.

So long as (k'_i, σ'_i) and (k_i, σ_i) encode the same value, this will preserve correctness of HGate, and hence the garbling scheme.

Hash Compression. Unfortunately, directly applying this straightforward method of rekeying as described above removes any hope for succinctness, as the garbler will need to send the evaluator a key for every wire of the circuit. Instead, what we need is a *succinct* rekeying operation where a short rekeying key can re-encode many different wire encodings.

[3] [ILL25b] also describe an 'unlevelled' variant where $s = s'$ which relies on circular security which achieves depth-independent succinctness. Since our scheme will incur depth-dependent factors for other reasons, we will look at their leveled construction

One way to realize this is rather than attempting to rekeying individual wire values, we can instead rekey a short 'hash' of the wire values. For this to be useful with prior gate evaluation procedures, which operates directly on encodings of wire values, there needs to exist non-interactive procedures for both converting an encoding of wires to an encoding of the hash of said wires as well as the reverse. A bit more formally, our hashing scheme should support the following protocols[4]:

- $\mathsf{Compress}(\llbracket x \rrbracket) \to \llbracket \mathsf{Hash}(x) \rrbracket$: Given an encoding of x, which we recall takes the form $\{k_i\}$ for the garbler and $\{k_i + s \cdot w_i\}$ for the evaluator, they should respectively be able to compute shares k_{Hash} and $(k_{\mathsf{Hash}} + s \cdot \mathsf{Hash}(\{w_i\}))$.
- $\mathsf{Expand}(\llbracket \mathsf{Hash}(x) \rrbracket) \to \llbracket x \rrbracket$: Conversely, given shares k_{Hash} and $(k_{\mathsf{Hash}} + s \cdot \mathsf{Hash}(\{w_i\}))$, the garbler and evaluator should each compute shares $\{k_i\}$, $\{k_i + s' \cdot w_i\}$ respectively.

This enables the garbler and evaluator to alternate between encodings of the wires (to compute HGate) and encodings of hash of wires (to run Rekey).

It turns out by virtue of the additive structure of the encodings, any linear hash function $\mathsf{Hash}(\mathbf{x}) = \mathbf{x}^\intercal \cdot \mathbf{u}$ can be 'compressed' while supporting the share format. Thus, the main challenge will be supporting the Expand operation. Much like how succinct gate evaluation HGate is constructed in prior works, this will rely on a publishing some cryptographic 'hints' dependent on garbling secrets s, s' which allow the garbler and evaluator to compute $\llbracket x \rrbracket$ from $\llbracket \mathsf{Hash}(x) \rrbracket$ while simultaneously hiding s, s'. Inspired by the hashing and reconstruction technique recently introduced by [AMR25, Wee25] for constructing succinct-time RAM garbling and ABE with succinct ciphertext from lattices, given coefficients $\mathbf{u} = (u_1, \ldots, u_N)^\intercal$ to a linear hash, we can convert an encoding of $\mathsf{Hash}(\mathbf{x}) = \mathbf{x}^\intercal \cdot \mathbf{u}$ into an encoding of $\mathbf{x}$ by publishing a random vector $\mathbf{t} = (t_1, \ldots, t_N)^\intercal$ as well as the matrix $\mathbf{W} = s \cdot \mathbf{t} \cdot \mathbf{u}^\intercal - s' \cdot \mathbf{I}_N$, embedding s' in the diagonal of $\mathbf{W}$ masked by term $s \cdot t_i \cdot u_i$. Now, given a sharing of the form $(k, k + s \cdot \mathsf{Hash}(\mathbf{x}))$, the garbler and evaluator can recover a share of x_i by first multiplying their respective sharings by t_i

$$t_i \cdot \llbracket \mathsf{Hash}(\mathbf{x}) \rrbracket_G = t_i \cdot k \quad , \quad t_i \cdot \llbracket \mathsf{Hash}(\mathbf{x}) \rrbracket_E = t_i \cdot (k + s \cdot (\mathbf{u}^\intercal \cdot \mathbf{x})) = t_i \cdot k + s \cdot t_i \cdot \mathbf{u}^\intercal \cdot \mathbf{x}.$$

This latter term of the evaluator's encoding is in fact exactly the ith row of $\mathbf{W}$ multiplied by $\mathbf{x}$, which allows the evaluator to compute

$$t_i \cdot \llbracket \mathsf{Hash}(\mathbf{x}) \rrbracket_E - \langle \mathbf{w}_i, \mathbf{x} \rangle = t_i \cdot k + s \cdot t_i \cdot \mathbf{u}^\intercal \cdot \mathbf{x} - \left(s \cdot (t_i \cdot \mathbf{u}^\intercal) \cdot \mathbf{x} - s' \cdot \mathbf{e}_i^\intercal \cdot \mathbf{x}\right) = t_i \cdot k + s' \cdot x_i.$$

where $\mathbf{e}_i$ is the ith canonical basis vector. The garbler and evaluator can now use

$$t_i \cdot k = \llbracket x_i \rrbracket_G, t_i \cdot k + s' \cdot x_i = \llbracket x_i \rrbracket_E$$

as their new encoding of x_i.

While this gives us the desired functionality, directly giving out $\mathbf{W}$ clearly compromises any sort of secrecy of s, s'. Instead, we will need to give out cryptographic 'encryption' of $\mathbf{W}$ which still enables us to perform the above computation. We discuss how this is instantiated in various contexts below:

Group-based Construction: In both prime-order and Paillier groups, the high level idea is instead of publishing $\mathbf{t}, \mathbf{W}$, the garbler instead publishes $g^{\mathbf{t}}, g^{\mathbf{W}}$. Intuitively, this allows us to use the secrecy of these terms in the exponent to act as an ElGamal-esque encryption of the secret s'. Since the garbler and evaluator only perform linear operations, this can be directly translated to computing $(g^{\mathbf{t}})^{\llbracket \mathsf{Hash}(\mathbf{x}) \rrbracket}$, $(g^{\mathbf{W}})^{\mathbf{x}}$, which gives an appropriate additive sharing *in the exponent of* $\mathbb{G}$. Since rekeying is a linear operation, this can be done directly in the exponent of the group! However, while such sharings in the exponent of the group are sufficient for rekeying, such an encoding is insufficient for HGate. Thus, our Expand operation will need to recover a sharing over $\mathbb{Z}$, which we achieve using the techniques of distributed discrete log DDLog protocol [BGI16]. This technique comes with two caveats (in prime-order groups): it incurs some inverse polynomial correctness error, and only works if the additive shares have some bounded polynomial difference. The former will simply reflect as an inverse polynomial correctness error in our 'base' garbling protocol, which can be amplified through generic techniques. With respect to the latter, notice while we can always

[4]In our real scheme, we refer to Expand as HRead which also has the syntax to 'locally' extract/read a single wire value from the hash.

maintain our initial input $\mathbf{x}$ and the shares of the input encoding $[\![x]\!]$, have bounded distance, the term of $s' \cdot \mathbf{x}$ in $g^{\mathbf{W}}$ could be as large as the group order p itself. However, rather than giving out encryptions of s' directly, we can instead bit-decompose s', using $\mathbf{W}$ to encode the bits of s' which are only 'repacked' after the DDLog returns the encoding to $\mathbb{Z}$.

At a glance, it may not be clear that this hashing has accomplished anything, as although the hash itself is now shorter than the original input, the garbler also has to give out $\mathsf{ct}_{s,s'} = g^{\mathbf{t}}, g^{\mathbf{W}}$, which are at least as long as the input of Hash (and in fact quadratically longer than said input). However, an important distinction between these two quantities is that $\mathsf{ct}_{s,s'}$ are *reusable* (within the same circuit depth of a garbling scheme). Thus, by fixing a compression factor K our garbling scheme pays a one-time cost of $K^2 \cdot D$ to reduce the re-keying communication by a factor of K. We can see this can be asymptotically minimized by setting $K = \left(\frac{|C|}{D}\right)^{\frac{1}{3}}$, incurring $D + |C|^{\frac{2}{3}} D^{\frac{1}{3}}$ total communication (ignoring $\mathsf{poly}(\lambda)$ factors. We also note that by assuming a circular variant of power-DDH, we can reuse the public parameters across layers, incurring a one-time cost of K^2 and rather than $K^2 \cdot D$, meaning our total communication can be brought down to $D + |C|^{\frac{2}{3}}$.

Lattice-based Construction In the setting of lattices, this same hashing idea can be executed by simply publishing $\mathbf{t}, \mathbf{u}$ in the clear and an "encryption" as simply a noisy version of $\mathbf{W} + \varepsilon$. In fact, as observed in prior work [AMR25, Wee25], since these hash coefficients can be published in the clear, this enables a recursive Merkle-style construction where the output of the hash itself is then again taken as input to the compressing hash[5]. This enables the hashing to be instantiated with a *fixed* compression factor[6] independent of the circuit size, meaning unlike in the group-based setting, the size of the rekeying communication can be made independent of $|C|$ (modulo the depth dependence). This recursion introduces one additional constraint - since the output of Hash will need to be taken as input to itself, the hashing must also preserve (up to bounded polynomial factors) the distance of the additive sharing in order for us to apply distributed rounding. In other words, the coefficient $\mathbf{u}$ of Hash must be small. We can accomplish this by once again relying on bit-decomposition, this time using $\mathsf{Bits}(\mathbf{u})$ rather than $\mathbf{u}$ as the coefficients to hash, which allows us to bound the norm of $||\mathsf{Hash}(\mathbf{x})||$ relative to $||\mathbf{x}||$. Since bit 'recomposition' too is a linear operation (multiplying by $[1, 2, \ldots 2^{\log q}]$), this once again preserves the share format described above.
Putting these ideas together, our overall garbling protocol proceeds layer-by-layer over a circuit C as follows:

- On layer j of the circuit, the garbler and evaluator begin with shares $k^{(j)}_{\mathsf{Hash}}, k^{(j)}_{\mathsf{Hash}} + s \cdot \mathsf{Hash}\left(\{w_i^{(j)}\}\right)$, where $k^{(j)}_{\mathsf{Hash}} \leftarrow \mathsf{RekeySamp}(1^\lambda)$ is independently sampled.
- The garbler and evaluator run Expand to obtain shares of the form $\{k^{(j)}\}, \{k^{(j)} + s' \cdot w_i^{(j)}\}$.
- The garbler and evaluator use HGate to compute the outputs of the jth layer of the circuit $\{k^{(j+1)}\}, \{k^{(j+1)} + s'' \cdot w_i^{(j+1)}\}$.
- The wire shares are then hashed down via Compress to a succinct $\left(k'^{(j+1)}_{\mathsf{Hash}}, k'^{(j+1)}_{\mathsf{Hash}} + s'' \cdot \mathsf{Hash}\left(\{w_i^{(j+1)}\}\right)\right)$.
- The garbler sends the evaluator $\left(k^{(j+1)}_{\mathsf{Hash}} - k'^{(j+1)}_{\mathsf{Hash}}\right)$, the difference between the garbler's current share of the hash and a freshly sampled one.

Since both the hash key and rekeying material are sublinear in $|C|$, this yields a succinct and low depth garbling scheme. In the setting of private garbling, this communication is dominated by the 1-bit-per-gate cost of HSS communication.

[5] In group instantiations described above, the coefficients $\mathbf{t}, \mathbf{u}$ must be hidden and hence so are the hash values $\mathsf{Hash}(\mathbf{x}) = \mathbf{u}^\intercal \cdot \mathbf{x}$. However, to apply recursive hashing, the evaluator must know $\mathsf{Hash}(\mathbf{x})$ in the clear.

[6] Prior work generally instantiates this compression factor as a constant such as two. However, for our entire garbling computation to remain in NC^1, our full constructions will require *constant* recursive depth, the compression factor is set proportional to security parameter λ.

Reusable Garbling. In the setting of reusable garbling [GKP+13], the garbler can produce a single garbled circuit which can be reused between multiple sets of garbled inputs. Ultimately, we can view the instantiation of reusable garbling through the homomorphic lattice encodings of [BGG+14] as an *approximate* additive sharing of $\mathbf{s}^\intercal \cdot x \otimes \mathbf{G}$ where the garbler holds $\mathbf{s}^\intercal, \mathbf{B}$ (which can be used to compute $\mathbf{s}^\intercal \cdot \mathbf{B}$) and the evaluator holds share $\mathbf{s}^\intercal(\mathbf{B} - x \otimes \mathbf{G}) + \mathbf{e}^\intercal$ with garbling secret $\mathbf{s}$. The homomorphic evaluation procedures of [BGG+14, BTVW17] directly enable the independent derivation from

$$\mathbf{B} \mapsto \mathbf{B}_C \quad , \quad \mathbf{s}^\intercal(\mathbf{B} - x \otimes \mathbf{G}) + \mathbf{e}^\intercal \mapsto \mathbf{s}^\intercal(\mathbf{B}_C - C(x) \otimes \mathbf{G}) + \mathbf{e}'^\intercal$$

for a circuit C. In fact, this derivation exactly corresponds to the garbler and evaluator's respective computation of HGate in the context of privacy-free garbling (and the same computation composed under a layer of FHE in the setting of private garbling). However, just like the techniques for gate evaluation in the non-reusable setting, the depth of both sides of this evaluation procedure grow with the circuit it is evaluating.

We can use the same high-level approach of rekeying and hash compression to break this symmetry in computation depth while preserving succinctness. However, the encoding of [BGG+14] puts two additional constraints on our techniques - (1) for the correctness of [BGG+14] homomorphic evaluation, the garbler and evaluator's share now *both* depend on a secret $\mathbf{s}$ and (2) the reusability of this garbling scheme relies on the garbler sampling a *new* $\mathbf{s}$ for each garbled input. Thus, the garbled circuit itself cannot be dependent on $\mathbf{s}$.

While our hashing construction described above suffers from both of these issues (the ciphertexts published for Hash depend on secret $\mathbf{s}$, and the Expand step produces an encoding under a different $\mathbf{s}'$, which would no longer be a [BGG+14] encoding), we can actually directly use the key-homomorphic RAM hash of [AMR25] as the natural [BGG+14]-compatible analogue of our Hash protocol. This comes at the expense of relying on a stronger assumption (decomposed LWE), but otherwise retains the same structure.

We also need to modify our rekeying protocol to respect these constraints. Recall in the non-reusable setting described above, a garbler can 'rekey' an evaluator's share by simply sending them the difference between his current share k' and his freshly sampled share k. This would mean 'rekey' corresponds to $(\mathbf{s}^\intercal \cdot \mathbf{B}_{\mathsf{Hash}} - \mathbf{s}^\intercal \cdot \mathbf{B}'_{\mathsf{Hash}})$, which of course is dependent on $\mathbf{s}$. To avoid this, the garbler instead publishes an additional fresh matrix $\mathbf{A}$ in the garbled circuit which it has sampled with a short trapdoor. The rekeying key is now instead computed as a short preimage of the form $\mathbf{A}^{-1}(\mathbf{B}_{\mathsf{Hash}} - \mathbf{B}'_{\mathsf{Hash}})$. When it comes time to garble an input x, the garbler also sends $\mathbf{s}^\intercal\mathbf{A} + \mathbf{e}^\intercal$, which, when multiplied by rekeying key, recovers a noisy $(\mathbf{s}^\intercal \cdot \mathbf{B}_{\mathsf{Hash}} - \mathbf{s}^\intercal \cdot \mathbf{B}'_{\mathsf{Hash}})$.

With these modifications, however we are able to obtain a succinct NC^1 garbling scheme from decomposed LWE. The same techniques also naturally extend to the setting of attribute-based encryption with succinct and NC^1 key generation.

3 Definitions

Notation We will refer to (layered) circuit $C : \{0,1\}^\ell \to \{0,1\}$ computable by a poly-size and depth d circuit as a series of gates, indexed by a tuple $(i, j) \in \mathbb{N}^2$ corresponding to the depth and number of that particular gate. Each gate $g_{(i,j)}$ is of the form $(\mathsf{op}, \mathsf{lw}, \mathsf{rw})$, where $\mathsf{op} \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}\}$ and lw, rw are the index tuples corresponding to the left and right input wires respectively. We will use C_i to denote the collection of depth i gates and $C_{[i]}(x)$ to denote the circuit which computes all depth $\leq i$ wires of C on x.

Definition 3.1 (Garbled Circuits)**.** Let λ be a security parameter. A garbling scheme is a tuple of efficient randomized algorithms $\Pi_{\mathsf{garble}} = (\mathsf{Setup}, \mathsf{GarbC}, \mathsf{GarbInp}, \mathsf{Eval})$:

- $\mathsf{Setup}(1^\lambda, 1^d) \to \mathsf{pp}$: On input security parameter λ and circuit depth d, the Setup algorithm outputs public parameters pp.
- $\mathsf{GarbC}(\mathsf{pp}, C) \to (\hat{C}, \mathsf{gk})$: On input public parameters pp and circuit C (of depth $\leq d$), the GarbC algorithm outputs garbled circuit $\hat{C}$ and garbling key gk.
- $\mathsf{GarbInp}(\mathsf{gk}, x) \to \hat{x}$: On input garbling key gk and input x, the GarbInp algorithm outputs a garbled input $\hat{x}$.
- $\mathsf{Eval}(C, \hat{C}, \hat{x}) \to \{0,1\}$: On input circuit C, garbled circuit $\hat{C}$ and garbled input $\hat{x}$, the Eval algorithm computes the evaluation of the garbling circuit.

We will consider the following syntactic and semantic properties for a garbling scheme:

- **Correctness:** For all polynomials $p(\cdot)$, $d(\cdot)$, there exists negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all circuits $C : \{0,1\}^\ell \to \{0,1\}$ of size $|C| \leq p(\lambda)$ and depth $\leq d(\lambda)$, all inputs $x \in \{0,1\}^\ell$,

$$\Pr\left[\mathsf{Eval}(C, \hat{C}, \hat{x}) = C(x) \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d) \\ (\mathsf{gk}, \hat{C}) \leftarrow \mathsf{GarbC}(\mathsf{pp}, C) \\ \hat{x} \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

- **Decomposability:** We say a garbling scheme Π_{garble} is decomposable if every bit of the input garbling $\hat{x}$ depends on at most one bit of x.

- **Succinctness:** We say a garbling scheme Π_{garble} is *succinct* if there exists a polynomial p such that for every $\lambda, d \in \mathbb{N}$, circuits C with depth d, input $x \in \{0,1\}^\ell$, and size $|C| > p(\lambda, d, \ell)$, the garbling size satisfy $|\hat{C}| + |\hat{x}| + |\mathsf{dk}| < |C| \cdot \log |C|$.

  We say a garbling scheme Π_{garble} is *depth-succinct* if there exists a polynomial p' such that for every $\lambda, d \in \mathbb{N}$, circuits C with depth d, input $x \in \{0,1\}^\ell$, the garbling size satisfy $|\hat{C}| + |\hat{x}| + |\mathsf{dk}| < p'(\lambda, d, \ell)$.

  We note that one could consider even stronger definitions of succinctness (and general garbling) which does not a-priori bound or depend on the depth of the circuit. However, our constructions will only consider this depth-dependent case.

We will consider two different distinct 'flavors' of garbled circuits security, each of which will require some additional syntax and algorithms to describe. We will also consider both security definitions in the context of non-reusable and reusable garbled circuits, where changes exclusive to the reusable setting are highlighted in green.

- **Input-Hiding Simulation:** In the traditional notion of garbled circuits, we require the garbled circuit to 'hide' the input it was evaluated on. To capture this notion, a garbling scheme will also be associated with a tuple of (efficient) simulator algorithms:

  - $\mathsf{Sim}_0(1^\lambda, 1^d) \to (\mathsf{pp}, \mathsf{st}_0)$: On input security parameter λ and depth bound d, the zeroth simulator outputs public parameters pp and state st_0.
  - $\mathsf{Sim}_1(\mathsf{st}_0, C) \to (\hat{C}, \mathsf{gk}, \mathsf{st}_1)$: On input security prior simulator state st_0 and circuit C, the first simulator outputs garbled circuit $\hat{C}$, the garbling key gk (in the reusable case) and new simulator state st_1.
  - $\mathsf{Sim}_2(\mathsf{st}_1, b) \to \hat{x}$: On input simulator state st_1 and circuit output bit b, the second simulator outputs a garbled input $\hat{x}$.

  For an adversary $\mathcal{A}$, define the following as the (selective) input-hiding simulation security experiment $\mathsf{ExptSS}^{(b)}_{\mathcal{A}}(\lambda)$ as follows:

  1. The adversary specifies a depth bound 1^d, a circuit C (of depth $\leq d$), and {one, arbitrarily many} inputs x. The challenger responds as follows:
     - If $b = 0$, the challenger computes the real garbling scheme:
       (a) The challenger samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$.
       (b) The challenger samples garbled circuit $(\hat{C}, \mathsf{gk}) \leftarrow \mathsf{GarbC}(\mathsf{pp}, C)$.
       (c) The challenger samples garbled input(s) $\hat{x} \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x)$ for each x submitted by the adversary.
     - If $b = 1$, the challenger computes the simulated garbling scheme:
       (a) The challenger samples public parameters $(\mathsf{pp}, \mathsf{st}_0) \leftarrow \mathsf{Sim}_0(1^\lambda, 1^d)$.
       (b) the challenger samples garbled circuit $(\hat{C}, \mathsf{gk}, \mathsf{st}_1) \leftarrow \mathsf{Sim}_1(\mathsf{st}_0, C)$.
       (c) The challenger samples garbled input(s) $\hat{x} \leftarrow \mathsf{Sim}_2(\mathsf{st}_1, C(x))$ for each x submitted by the adversary.

2. The challenger returns $(\mathsf{pp}, \hat{C}, \mathsf{gk}, \{\hat{x}\})$.
3. The adversary outputs a bit b', which is the output of the experiment.

We note that in the reusable setting, the adversary gets two additional 'powers' - it can see the garbling key gk and request as many garbled input labels as desired, while in the non-reusable setting, an adversary makes exactly one such query. We say a scheme Π_{garble} is input-hiding simulation secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathsf{ExptSS}^{(0)}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{ExptSS}^{(1)}_{\mathcal{A}}(\lambda) = 1]\right| = \mathsf{negl}(\lambda).$$

- **Authenticity:** An orthogonal property of garbled circuits that could be desirable even in the absence of input hiding is the notion of label *authenticity*. This is the main security property of interest in the setting of privacy-free garbling [FNO15] where the garbled circuit instead simply represents a way to authenticate computation. To capture this requirement, we additionally assume our garbling algorithms have the following structure:
  - $\mathsf{GarbInp}(\mathsf{gk}, x) \to \hat{x} = (L_x, \mathsf{dk}), \mathsf{auk}$: In the input garbling algorithm, let the garbled input $\hat{x}$ be decomposed into two part - a garbled input label L_x and a decoding key dk. In addition, GarbInp can implicitly compute a authentication key auk.
  - $\mathsf{Eval}(C, \hat{C}, \hat{x}) \to \{0,1\}$: The evaluation algorithm can then be decomposed into two steps:
    * $\mathsf{EvalLabel}(C, \hat{C}, L_x) \to L_{\mathsf{out}}$: First, the EvalLabel algorithm takes the garbled circuit $\hat{C}$ as well as the garbled input labels L_x and computes an output label L_{out}.
    * $\mathsf{DecodeLabel}(\mathsf{dk}, L_{\mathsf{out}}) \to \{0,1\} \cup \perp$: Once the output label is computed, a second DecodeLabel algorithm applies the decoding key dk to the label, which outputs a bit (which is also the output of Eval).
  - $\mathsf{Authenticate}(\mathsf{auk}, L_{\mathsf{out}}) \to \{0,1\}$: Privacy-free garbling is also equipped with an 'authentication' algorithm, which takes in authentication key auk and a label L_{out} and outputs a bit, which is 1 on any label generated by the honest execution of GarbInp, EvalLabel.

Now we can define the (selective) authenticity game of a garbling scheme with respect adversary $\mathcal{A}$ through the following security $\mathsf{ExptAuth}_{\mathcal{A}}(\lambda)$:

1. The adversary specifies a depth bound 1^d, a circuit C (of depth $\leq d$), and one input x^*. The challenger computes the following:
   (a) The challenger samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$.
   (b) The challenger samples garbled circuit $(\hat{C}, \mathsf{gk}) \leftarrow \mathsf{GarbC}(\mathsf{pp}, C)$.
   (c) The challenger samples garbled input $\hat{x^*} \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x^*)$, as well as accompanying authentication key auk^*.
2. The challenger returns $(\mathsf{pp}, \hat{C}, \mathsf{gk}, \hat{x^*} = L_{x^*}, \mathsf{dk}^*)$.
3. The adversary submits a label L^*. The output of the experiment is 1 iff $1 - C(x^*) = \mathsf{DecodeLabel}(\mathsf{dk}^*, L^*)$ and $\mathsf{Authenticate}(\mathsf{auk}^*, L^*) = 1$.

As in the input-hiding simulation security, the setting of reusable authenticity enables the adversary to see the garbling key gk. As a result, they can generate as many garbled inputs it desires. In either case, we say a scheme Π_{garble} satisfies authenticity if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{ExptAuth}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

3.1 Garbled Circuits with Algebraic Output Labels

Our constructions of private and non-private garbling will all conform to a similar high-level algebraic structure. Below, we describe this structure of our garbled circuits and some properties which are equivalent to the traditional simulation security/authenticity definitions in garbling, which is what our constructions will directly show.

Definition 3.2 (Algebraic Garbled Circuit)**.** We say a garbling scheme Π_{garble} with syntax as Definition 3.1 above is algebraic over a ring $\mathcal{R}$ if it takes the following form:

- **Algebraic Labeling:** The garbling scheme will be associated with an integer ring $\mathcal{R} = \mathbb{Z}$ or $\mathbb{Z}_q$ along with parameter n where $n \in \mathsf{poly}(\lambda)$, along with a distribution $\mathcal{D}^{(\mathcal{R})}$ over $\mathcal{R}^{n+1}$. We let $\mathcal{D}^{(\mathcal{R})}_{n+1}$ denote the marginal distribution of the last entry of $\mathcal{D}^{(\mathcal{R})}$. We require $\mathcal{D}^{(\mathcal{R})}_{n+1} = 0$ with only negligible probability. The garbling algorithms can be broken down as follows (inheriting the additional syntax and structure of the authenticity definition):
  - The input garbling algorithm $\mathsf{GarbInp}$ can be broken down into the following steps:
    - Sample $\mathbf{s}^{(\mathsf{out})} \xleftarrow{R} \mathcal{D}$.
    - $\mathsf{GarbInp}'(\mathsf{gk}, x, \mathbf{s}^{(\mathsf{out})})$ returns output 'tag' $\mathbf{r}^{(\mathsf{out})}$ and garbled input labels L_x.
    - Set decoding key dk as $r^{(\mathsf{out})}_{n+1}$, the last entry of $\mathbf{r}^{(\mathsf{out})}$, and authentication key auk as vectors $(\mathbf{r}^{(\mathsf{out})}, \mathbf{s}^{(\mathsf{out})})$.
  - Decomposing Eval into an $\mathsf{EvalLabel}$ which computes a label of the form

    $$L_{\mathsf{out}} = \mathbf{r}^{(\mathsf{out})} + \mathbf{s}^{(\mathsf{out})} \cdot C(x)$$

    where and a decoding step. Recall $\mathsf{dk} = r^{(\mathsf{out})}_n$, so decoding can proceed by simply computing

    $$\mathsf{dk} - L_{\mathsf{out},n+1} = r^{(\mathsf{out})}_{n+1} - r^{(\mathsf{out})}_{n+1} - s^{(\mathsf{out})}_{n+1} \cdot C(x).$$

    Since $s^{(\mathsf{out})}_{n+1}$ is nonzero with overwhelming probability, the Eval outputs 0 iff this quantity is 0.
  - The $\mathsf{Authenticate}$ algorithm, which takes in $\mathsf{auk} = (\mathbf{r}^{(\mathsf{out})}, \mathbf{s}^{(\mathsf{out})})$, returns 1 on label L if $L = \mathbf{r}^{(\mathsf{out})}$ or $L = \mathbf{r}^{(\mathsf{out})} + \mathbf{s}^{(\mathsf{out})}$.
- **One-Wayness:** We define the one-wayness as the algebraic analogue to authenticity. The one-wayness experiment for garbled circuits ExptOW proceeds as follows:
  1. The adversary specifies a depth bound 1^d, a circuit C (of depth $\leq d$), and an input x^*. The challenger computes the following:
     (a) The challenger samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$.
     (b) The challenger samples garbled circuit $(\hat{C}, \mathsf{gk}) \leftarrow \mathsf{GarbC}(\mathsf{pp}, C)$.
     (c) The challenger samples garbled input $\hat{x^*} \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x^*)$, as well as accompanying authentication key auk^* using secret randomness $\mathbf{s}^*$.
  2. The challenger returns $(\mathsf{pp}, \hat{C}, \mathsf{gk}, \hat{x^*})$.
  3. The adversary submits a guess $\mathbf{s}'$. The output of the experiment is 1 iff $\mathbf{s}' = \mathbf{s}^*$.

  We say a scheme Π_{garble} satisfies one-wayness if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

  $$\Pr[\mathsf{ExptOW}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda).$$
- **Semantic Security:** We define the (selective) semantic security experiment for garbled circuits $\mathsf{ExptIND}^{(b)}$ as follows. This property has also previously been referred to as obliviousness [BHR12]:
  1. The adversary specifies a depth bound 1^d, a circuit C (of depth $\leq d$), and a pair of inputs x_0, x_1. The challenger computes the following:

(a) The challenger samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$.
(b) The challenger samples garbled circuit $(\hat{C}, \mathsf{gk}) \leftarrow \mathsf{GarbC}(\mathsf{pp}, C)$.
(c) The challenger samples garbled input $\hat{x} \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x_b)$, which it parses as $\hat{x} = (L_x, \mathsf{dk})$.

2. The challenger returns $(\mathsf{pp}, \hat{C}, \mathsf{gk}, L_x)$.
3. The adversary submits a bit b' which is the output of the experiment.

We say a scheme Π_{garble} satisfies semantic security if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathsf{ExptIND}^{(1)}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{ExptIND}^{(0)}_{\mathcal{A}}(\lambda) = 1]\right| = \mathsf{negl}(\lambda).$$

It is easy to see that any garbling scheme satisfying the structure of algebraic labeling will naturally satisfy correctness. We show below that the normal garbling properties of authenticity and input-hiding simulation security also follow from one-wayness and semantic security respectively in this context. Our reductions below follow the adaptive versions of the definitions, but the selective reductions of one-wayness to authenticity and semantic security to simulation security follow analogously.

Lemma 3.3. *Suppose an algebraic garbling scheme Π_{garble} satisfies one-wayness. Then it is also satisfies authenticity as a privacy-free garbling scheme.*

Proof. Suppose $\mathcal{A}$ is an adversary against the one-wayness of Π_{garble} which succeeds with probability ε. Then consider the following reduction $\mathcal{B}$ against the authenticity of Π_{garble}:

0. Reduction $\mathcal{B}$ begins running $\mathcal{A}$.
1. Adversary $\mathcal{A}$ submits a depth bound 1^d a circuit C, and an input x^*, which $\mathcal{B}$ simply forwards to the challenger.
2. Reduction $\mathcal{B}$ receives $(\mathsf{pp}, \hat{C}, \mathsf{gk}, \hat{x^*} = (L_{x^*}, \mathsf{dk}_{x^*}))$, which it simply forwards to $\mathcal{A}$.
3. Adversary $\mathcal{A}$ submits a label $L_{1-C(x^*)}$. Reduction $\mathcal{B}$ computes label $L_{C(x^*)} \leftarrow \mathsf{EvalLabel}(C, \hat{C}, L_{x^*})$ and returns $\mathbf{s}' = L_1 - L_0$.

Recall for $\mathcal{A}$ to succeed, adversary $\mathcal{A}$ must submit a label L which which both authenticates and decodes to $1 - C(x)$. For the former to be true, $L = \mathbf{r}^{(\mathsf{out})} + b \cdot \mathbf{s}^*$ for $b \in \{0, 1\}$. This can be decomposed into two cases:

- $L = \mathbf{r}^{(\mathsf{out})} + C(x) \cdot \mathbf{s}^*$: In this case, recall that since $\mathsf{dk} = r_n^{(\mathsf{out})}$, this will decode to $C(x)$, and the adversary did not win the authenticity game.
- $L = \mathbf{r}^{(\mathsf{out})} + (1 - C(x)) \cdot \mathbf{s}^*$: In this case, recall that by definition the output L_{out} of $\mathsf{EvalLabel}$ is $\mathbf{r}^{(\mathsf{out})} + C(x) \cdot \mathbf{s}^*$. Thus we can compute $\mathbf{s}^*$ simply as their difference, and our reduction $\mathcal{B}$ wins the one-wayness game.

□

Lemma 3.4. *Suppose an algebraic garbling scheme Π_{garble} satisfies semantic security. Then it is also satisfies input-hiding simulation security as a private garbling scheme.*

Proof. We define the private garbling simulators $\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2$ as follows:

- $\mathsf{Sim}_0(1^\lambda) \to (\mathsf{pp}, \mathsf{st}_0)$: On input security parameter λ, the zeroth simulator simply runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and sets both outputs to pp.
- $\mathsf{Sim}_1(\mathsf{st}_0 = \mathsf{pp}, C) \to (\hat{C}, \mathsf{gk}, \mathsf{st}_1)$: On input security parameter λ and circuit C, the first simulator simply runs $(\hat{C}, \mathsf{gk}) \leftarrow \mathsf{GarbC}(\mathsf{pp}, C)$, and outputs $(\hat{C} = \hat{C}, \mathsf{gk} = \mathsf{gk}, \mathsf{st}_1 = \mathsf{gk})$.
- $\mathsf{Sim}_2(\mathsf{st}_1, C(x)) \to (L_x, \mathsf{dk})$: On input trapdoor $\mathsf{td} = \mathsf{gk}$ and circuit output $C(x)$, the second simulator samples $s_{n+1}^{(\mathsf{out})} \xleftarrow{R} \mathcal{D}_{n+1}^{(\mathcal{R})}$, runs $\mathsf{GarbInp}(\mathsf{gk}, 0) \to L_0$, and outputs $L_x = L_0$ and $\mathsf{dk} = L_{0,n+1} - s_{n+1}^{(\mathsf{out})} \cdot C(x)$, where $L_{0,n+1}$ is the last entry of L_0.

Now, we can argue simulation security through a sequence of hybrids:

- Hyb_0 : This is the original input-hiding simulation security experiment when $b = 0$.

  1. The adversary specifies a depth bound 1^d, a circuit C (of depth $\leq d$), and {one, arbitrarily many} inputs x. The challenger responds as follows:
     (a) The challenger samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$.
     (b) The challenger samples garbled circuit $(\hat{C}, \mathsf{gk}) \leftarrow \mathsf{GarbC}(\mathsf{pp}, C)$.
     (c) The challenger samples garbled input(s) $\hat{x} = (L_x, \mathsf{dk}_x) \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x)$ for each x submitted by the adversary.
  2. The challenger returns $(\mathsf{pp}, \hat{C}, \mathsf{gk}, \{\hat{x}\})$.
  3. The adversary outputs a bit b', which is the output of the experiment.

- Hyb_1 : In this experiment, we sample switch to computing $\mathsf{dk} = L_{x,n+1} - C(x)$.

  1.(c) The challenger samples garbled input(s) $\hat{x} = (L_x, \mathsf{dk}_x) \leftarrow \mathsf{GarbInp}(\mathsf{gk}, x)$ and samples $s_{n+1}^{(\mathsf{out})} \xleftarrow{R} \mathcal{D}_{n+1}^{(\mathcal{R})}$ to set $\mathsf{dk}'_x = L_{x,n+1} - s_{n+1}^{(\mathsf{out})} \cdot C(x)$ for each x submitted by the adversary.
  2. The challenger returns $(\mathsf{pp}, \hat{C}, \mathsf{gk}, \{(L_x, \mathsf{dk}'_x\})$.

- Hyb_2 : In this experiment, we switch to computing $L_x \leftarrow \mathsf{GarbInp}(\mathsf{gk}, 0)$.

  1.(c) The challenger samples garbled input(s) $\hat{x} = (L_x, \mathsf{dk}_x) \leftarrow \mathsf{GarbInp}(\mathsf{gk},0)$ and samples $s_{n+1}^{(\mathsf{out})} \xleftarrow{R} \mathcal{D}_{n+1}^{(\mathcal{R})}$ to set $\mathsf{dk}'_x = L_{x,n+1} - s_{n+1}^{(\mathsf{out})} \cdot C(x)$ for each x submitted by the adversary.

Claim 3.5. *For all adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$,*

$$\Pr\left[\mathsf{Hyb}_0(\mathcal{A}, \lambda) = 1\right] = \Pr\left[\mathsf{Hyb}_1(\mathcal{A}, \lambda) = 1\right].$$

Proof. By definition of algebraic labeling, these are sampled identically. □

Claim 3.6. *Suppose Π_{garble} is semantically secure. Then for all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr\left[\mathsf{Hyb}_1(\mathcal{A}, \lambda) = 1\right] - \Pr\left[\mathsf{Hyb}_2(\mathcal{A}, \lambda) = 1\right]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose an adversary $\mathcal{A}$ distinguishes Hyb_1 and Hyb_2 with noticeable probability ε. Then the following is a reduction $\mathcal{B}$ against the semantic security of Π_{garble}. For simplicity, we present the case where Π_{garble} is non-reusable, but this extends to the reusable garbling case through a simple hybrid argument.

0. Reduction $\mathcal{B}$ begins running $\mathcal{A}$.

1. Adversary $\mathcal{A}$ specifies a depth bound 1^d, a circuit C (of depth $\leq d$), and an input x^*. Reduction $\mathcal{B}$ forwards $1^d, C$, and inputs $x_0 = x^*, x_1 = x$ to the challenger. $\mathcal{B}$ then samples $s_{n+1}^{(\mathsf{out})} \xleftarrow{R} \mathcal{D}_{n+1}^{(\mathcal{R})}$ and sets $\mathsf{dk}'_x = L_{x,n+1} - C(x)$, returning $(\mathsf{pp}, \hat{C}, \hat{x} = (L_x, \mathsf{dk}'_x))$ to the adversary.

2. The adversary submits a bit b' which is the output of the experiment.

We can see that when L_x is a garbled input label of x^*, the adversary's view is exactly that in Hyb_1, and similarly the when L_x is a garbled input label of 0 this is the adversary's view in Hyb_2. Thus the advantage of $\mathcal{B}$ in the semantic security game is exactly that of $\mathcal{A}$ in distinguishing $\mathsf{Hyb}_1, \mathsf{Hyb}_2$. □

Since Hyb_2 is exactly the adversary's view when receiving simulated garbled inputs per $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2)$ and depends only on the garbled input evaluation $C(x)$, these claims complete our proof of the lemma. □

4 A General Template for Low-Depth Succinct Garbling

We will take the view of [HK21, HKO22, ILL25b] presenting the $(\mathsf{GarbC}, \mathsf{GarbInp}, \mathsf{Eval})$ algorithms of a garbling scheme as a two-party protocol between a garbler P_G and evaluator P_E, who jointly hold an encoding of input x and homomorphically compute an encoding of $C(x)$ with one directional communication from the garbler to evaluator (representing the output of GarbC, GarbInp). This protocol roughly can be decomposed into two algorithms Setup, Enc run by P_G, and a subprotocol HGate jointly run by P_G, P_E.

- $\mathsf{Setup}(1^\lambda, 1^d, 1^{|C|}) \to \mathsf{pp}$: The setup algorithm takes in security parameter λ, along with circuit parameters (in our case depth d and (in some instances) circuit size $|C|$), and generates public data pp, which should be sent to P_E and be accessible for all following algorithms and sub-protocols.
- $\mathsf{Enc}(x) \to [\![x]\!], \mathsf{pd}$: The encoding algorithm takes as input the public data pd and a value x and outputs a two-party encoding $[\![x]\!] = ([\![x]\!]_G, [\![x]\!]_E)$ of x, public data pd. The encoding $[\![x]\!]_E$ and public data pd should be sent to P_E. Like pp, we will assume the public data pd is implicitly taken as input and available to all following algorithms and sub-protocols.
- $\mathsf{HGate}(\mathsf{op}, [\![x]\!], [\![y]\!]) \to [\![\mathsf{op}(x, y)]\!]$: The homomorphic gate evaluation protocol takes as input a gate op and input encodings $[\![x]\!], [\![y]\!]$ (jointly held by P_G, P_E) to compute an encoding of the output encoding $[\![\mathsf{op}(x, y)]\!]$ (jointly held by P_G, P_E). In some instances (in particular when relying on homomorphic secret sharing) this could require garbler to evaluator communication.

Once an encoding of $y = C(x)$ has been computed, the shares y_G and y_E can roughly be thought of as the decoding key and output label respectively. To make this general garbling framework low-depth while still retaining succinctness, we can retain the prior encoding Enc and homomorphic evaluation HGate sub-protocols while making use of three additional sub-protocols, a rekeying operation Rekey and a homomorphic HRead, HCompress, the latter two of which are defined with respect to a homomorphic Hash function. These protocols have the following properties:

- $\mathsf{Rekey}(1^\lambda, [\![x]\!] = ([\![x]\!]_G, [\![x]\!]_E)) \to [\![x]\!]' = ([\![x]\!]'_G, [\![x]\!]'_E)$: The rekeying operation will take a joint encoding of a value x and transform it into a new *fresh* encoding of the same value x. Importantly, the garbler's new share of the encoding $[\![x]\!]'_G$ will be *independent* of the input. To capture this requirement, we will decompose the rekeying protocol into the following steps:
  - $\mathsf{RekeySamp}(1^\lambda) \to [\![x]\!]'_G$: The garbler locally samples a fresh share $[\![x]\!]'_G$.
  - $\mathsf{RekeyGen}([\![x]\!]_G, [\![x]\!]'_G) \to \mathsf{rk}$: Using the freshly generated share $[\![x]\!]'_G$, as well as the original input encoding $[\![x]\!]_G$, the garbler generates a rekeying 'hint' rk, which is sent to the evaluator.
  - $\mathsf{RekeyEval}([\![x]\!]_E, \mathsf{rk}) \to [\![x]\!]'_E$: The evaluator, using their prior encoding share $[\![x]\!]_E$ and the rekey rk, computes their new encoding $[\![x]\!]'_E$.

  Such a rekeying immediately enables a low depth garbler by allowing the garbler to generate their shares of each layer of the circuit C independently, then in parallel using HGate to evaluate the wires of each layer, before sending rekeying hints rk to the evaluator to 'bridge' the layers of the circuit.

  Unfortunately, in our realisations of such rekeying protocols, the length of rk, and hence the communication between P_G and P_E (which corresponds to the size of the garbled circuit), grows with the length of the value x which needs to be rekeyed. Since we will need to rekey every wire in our circuit, applying this idea naively results in a $\mathsf{poly}(\lambda) \cdot |C|$ overhead, losing any notion of succinctness in the resulting garbled circuit.
- $\mathsf{Hash}(x) \to h_x$: To compress the communication of our garbling protocol, instead of attempting to rekey the wires of circuit directly, we instead rekey a succinct *hash* of our wire labels, while simultaneously ensuring our hash retains enough structure to execute the layer-by-layer homomorphic computation on the underlying encoding.

– $\mathsf{HRead}(\llbracket h_x \rrbracket, i, P_E : \llbracket x \rrbracket_E) \to \llbracket x_i \rrbracket$: Using an encoding of the hash h_x, along with an index i, the parties can derive an encoding of the ith bit of x. To enable this sub-protocol on the evaluator's side, they will also need their share of the encoding of the original input $\llbracket x \rrbracket_E$. More concretely, the HRead step will require the evaluator to know (in the clear) the input to a linear hash function underlying Hash, which are simply wire values in the case of privacy-free garbling and one time pad encryptions of wire values in the case of private-garbling.

– $\mathsf{HCompress}(\llbracket x \rrbracket) \to (\llbracket h_x \rrbracket)$: Given encodings of input values $\llbracket x \rrbracket$, the parties can derive an encoding of h_x, the hash of string x, The computational depth of HCompress will depend *logarithmically* on the length of x.

Both HRead and HCompress will be able to be executed *noninteractively*.

Putting these together, we can realise a general framework for low-depth succinct garbling which will proceed as follows:
Inputs: P_G holds input x, circuit C, while P_E just holds circuit C.
Outputs: P_G outputs a decoding key dk, while P_E outputs a label L.

1. **Input Encoding Phase:** Party P_G samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d, 1^{|C|})$ and generates encoding and public data $(\llbracket x \rrbracket, \mathsf{pd}, \mathsf{st})$, where $\llbracket x \rrbracket = (\llbracket x \rrbracket_G, \llbracket x \rrbracket_E) \leftarrow \mathsf{Enc}(\mathsf{gk}, x)$, sending $\mathsf{pd}, \llbracket x \rrbracket_E$ to P_E.

2. **Homomorphic Evaluation:** The parties firstly jointly use HGate to evaluate the hash function Hash, obtaining an encoding $\llbracket h_x \rrbracket$. Then, dividing the circuit C into layers $C_1, \ldots, C_d$, P_G and P_E iteratively compute the following for $i \in [d]$:

   - We maintain the invariant that for all $i \in [d]$ computed thus far, $h^{(i)}$ is the hash of all of the output wires $w^{(i)}$ of the ith layer of circuit C, with $h_x = h^{(0)}$.
   - For each gate $j \in |C_i|$, the ith layer of circuit C, take gate $g_{(i,j)} = (\mathsf{op}_j, \mathsf{lw}_j = (\mathsf{ll}_j, \mathsf{li}_j), \mathsf{rw}_j = (\mathsf{rl}_j, \mathsf{ri}_j)) \in C_i$, where ll, rl indicates the "depth" of the left and right input wires, and li, ri indicate their indices. Then compute $\llbracket w^{(i)}_{\mathsf{lw}_j} \rrbracket \leftarrow \mathsf{HRead}(\llbracket h^{(\mathsf{ll}_j)} \rrbracket, \mathsf{li}_j, P_E : \llbracket w^{(\mathsf{ll}_j)} \rrbracket_E)$ and $\llbracket w^{(i)}_{\mathsf{rw}_j} \rrbracket \leftarrow \mathsf{HRead}(\llbracket h^{(\mathsf{rl}_j)} \rrbracket, \mathsf{ri}_j, P_E : \llbracket w^{(\mathsf{rl}_j)} \rrbracket_E)$.
   - For all gates $g_{(i,j)} \in C_i$, compute $\llbracket w^{(i)}_j \rrbracket \leftarrow \mathsf{HGate}(\mathsf{op}_j, \llbracket w^{(i)}_{\mathsf{lw}_j} \rrbracket, \llbracket w^{(i)}_{\mathsf{rw}_j} \rrbracket)$
   - Compute encoding $\llbracket h^{(i)} \rrbracket' \leftarrow \mathsf{HCompress}(\{\llbracket w^{(i)}_j \rrbracket\}_{j \in [|C_i|]})$.
   - Rekey encoding with $\mathsf{Rekey}(\llbracket h^{(i)} \rrbracket') \to \llbracket h^{(i)} \rrbracket$.

3. **Output Phase:** Let j_{out} be the index of the output gate. Derive output encoding $\llbracket y \rrbracket = (\llbracket y \rrbracket_G, \llbracket y \rrbracket_E) \leftarrow \mathsf{HRead}(\llbracket h^{(d)} \rrbracket, \llbracket w^{(d)} \rrbracket, j_{\text{out}})$. The garbler and evaluator derive dk, L from $\llbracket y \rrbracket_G$ and $\llbracket y \rrbracket_E$ respectively.

4.1 Garbling Depth

A garbling procedure following the above-outlined template naturally yields the following parallelization of the garbler's computation.

- **Input Encoding Phase:** Compute as above.
- **Homomorphic Evaluation:** Begin by computing $\llbracket h_x \rrbracket_G = \llbracket h^{(0)} \rrbracket_G$ as above. *In parallel,* for $i \in [d]$, perform the following:
  - Sample layer key $\mathsf{RekeySamp}(1^\lambda) \to \llbracket h^{(i)} \rrbracket_G$.
  - For all gates $g_j = (\mathsf{op}_j, \mathsf{lw}_j, \mathsf{rw}_j) \in C_i$, the garbler computes $\llbracket w^{(i)}_{\mathsf{lw}_j} \rrbracket_G \leftarrow \mathsf{HRead}(\llbracket h^{(\mathsf{ll}_j)} \rrbracket, \mathsf{li}_j)$ and $\llbracket w^{(i)}_{\mathsf{rw}_j} \rrbracket \leftarrow \mathsf{HRead}(\llbracket h^{(\mathsf{rl}_j)} \rrbracket, \mathsf{ri}_j)$[7].

[7]Recall that only the evaluator needs input encodings to execute HRead, while the garbler only needs their share of the hash and the index.

- For all gates $g_j \in C_i$, compute $[\![w_j^{(i)}]\!]_G \leftarrow \mathsf{HGate}(\mathsf{op}_j, [\![w_{\mathsf{lw}_j}^{(i)}]\!]_G, [\![w_{\mathsf{rw}_j}^{(i)}]\!]_G)$
- Compute encoding $[\![h^{(i)}]\!]'_G \leftarrow \mathsf{HCompress}(\{[\![w_j^{(i)}]\!]_G\}_{j\in[|C_i|]})$.

- In parallel for each $i \in [d]$, compute $\mathsf{RekeyGen}([\![h^{(i)}]\!]', [\![h^{(i)}]\!]) \to \mathsf{rk}_i$, which is (in the reusable case), output as part of gk.
- **Output Phase:** Compute as above.

We can see from this rearrangement that any garbling scheme which can be decomposed into our template will naturally have garbling depth independent of the circuit depth C, modulo any depth dependence incurred by the setup or encoding procedures, which we will show is independent or logarithmic for all of our constructions. In fact, in some instantiations, our succinct garbling schemes will be computable in depth $O(\log(\lambda, |C|))$, i.e. NC^1.

5 Non-Reusable Succinct Privacy-Free Garbling

In this section, we present our constructions of non-reusable succinct privacy-free garbling, which can be summarized in the following theorem:

Theorem 5.1 (Non-Reusable Succinct Privacy-Free Garbling under DDH)**.** *Assuming any of the following:*

- *DDH in prime-order groups that are subgroups of $\mathbb{Z}_p^*$ for some prime p;*
- *DDH in Paillier groups;*

There exists a succinct privacy-free garbling scheme (i.e., satisfying only authenticity in Definition 3.1) satisfying the following.

- *Garbling sizes of a circuit $C : \{0,1\}^\ell \to \{0,1\}$ with depth D satisfy:*

$$|\hat{C}| < (D + |C|^{2/3} \cdot D^{1/3}) \cdot \mathsf{poly}(\lambda),\ |\hat{x}| < \ell \cdot \mathsf{poly}(\lambda),\ |\mathsf{dk}| < \mathsf{poly}(\lambda).$$

- $\mathsf{GarbC}(\mathsf{pp}, C)$ *can be computed in* NC^1.

Theorem 5.2 (Non-Reusable Succinct Privacy-Free Garbling under RLWE)**.** *Assuming RLWE with respect to* $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ *as specified in Construction 5.28 under $B = \lambda^{\omega(1)}$, there exists a depth-succinct privacy-free garbling scheme satisfying the following.*

- *Garbling sizes of a circuit $C : \{0,1\}^\ell \to \{0,1\}$ with depth D satisfy:*

$$|\hat{C}| < D \cdot \mathsf{poly}(\lambda),\ |\hat{x}| < \ell \cdot \mathsf{poly}(\lambda),\ |\mathsf{dk}| < \mathsf{poly}(\lambda).$$

- $\mathsf{GarbC}(\mathsf{pp}, C)$ *can be computed in* NC^1.

In the following, we give preliminaries in Section 5.1, show the group instantiation in prime-order groups in Section 5.2 and show the lattice instantiation in Section 5.3. We omit the construction in Paillier groups, as it is analogous to that in prime-order groups.

5.1 Preliminaries

We consider two types of groups, prime-order groups and Paillier groups (of composite orders) and in this section. We provide a quick review of these groups and the standard DDH assumption in them.

Definition 5.3 (Prime-order Groups)**.** We consider prime-order groups defined by an instance generation algorithm Gen with the following syntax.

- $\mathsf{Gen}(1^\lambda)$ outputs (G, p, g) where G is a group description of prime order $p > 2^\lambda$, and g is a generator of G.

Definition 5.4 (DDH Assumption)**.** We say DDH holds in prime-order groups if the following holds:

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{DDH.Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{DDH.Gen}(1^\lambda), \\ a, b, c \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda .$$

We will use the distributed discrete log (DDLog) algorithms [BGI16, DKK18, RS21, OSY21] for prime-order groups and Paillier groups respectively.

Lemma 5.5 (Distributed Discrete Log for Prime-Order Groups [BGI16, DKK18])**.** *For a cyclic group G with order p and a generator g, an error bound $\delta \in (0, 1]$, a message bound B, there exists an algorithm $\mathsf{DDLog}^{\delta,B}_{G,g}$:*

- $\mathsf{DDLog}^{\delta,B}_{G,g}(\varphi : G \to \{0,1\}^{\lceil \log(2B/\delta) \rceil}, a \in G)$ *takes a function φ mapping group elements to bit strings, and an element a. It outputs a value $\alpha \in \mathbb{Z}_p$.*

The algorithm requires $O(\sqrt{B/\delta})$ group operations and evaluations of φ, and has the guarantee that for all $0 < \delta \leq 1$, $B < p$, $a \in G$, and $m \leq B$:

$$\Pr\left[\begin{array}{c} \mathsf{DDLog}^{\delta,B}_{G,g}(\varphi, a \cdot g^m) \\ = \mathsf{DDLog}^{\delta,B}_{G,g}(\varphi, a) + m \bmod p \end{array} \;\middle|\; \varphi \leftarrow \$ \right] \geq 1 - \delta,$$

where $\varphi \leftarrow \$$ means sampling at random from all possible mappings.

Remark 5.6. We note that the DDLog algorithm can be implemented in depth $O(\log \lambda)$, assuming $B/\delta < \mathsf{poly}(\lambda)$, φ can be evaluated in depth $O(\log \lambda)$. In our constructions using prime-order groups, we instantiate (pseudo)random mappings φ via a $\mathsf{PRF} : \{0,1\}^\lambda \times [2^\lambda] \times G \to \{0,1\}^\lambda$. In more detail, the parties hold a common random key $\leftarrow \{0,1\}^\lambda$, and maintain a counter $c \in [2^\lambda]$. Every invocation of DDLog is instantiated with $\varphi(\cdot) := \mathsf{PRF}(\mathsf{key}, c, \cdot)$ (with suitably truncated outputs) for distinct counter values c. Indeed, under the DDH assumption in prime-order subgroups of $\mathbb{Z}^*_{p'}$ for primes p', there exists such PRFs with depth $O(\log \lambda)$ [NR97].

Definition 5.7 (Paillier Groups)**.** Paillier groups are defined by the following instance generation algorithm Gen.

- $\mathsf{Gen}(1^\lambda, 1^\zeta)$ uniformly samples two λ-bit primes p, q such that $p = 2p' + 1$, $q = 2q' + 1$ where p', q' are also primes. It outputs $(N = pq, \zeta)$ as the group description of $G = \mathbb{Z}^*_{N^{\zeta+1}}$.

Lemma 5.8 (Facts about Paillier Groups [Pai99, DJ01])**.** *Let $G = \mathbb{Z}^*_{N^{\zeta+1}}$ be a Paillier group sampled by $\mathsf{Gen}(1^\lambda, 1^\zeta)$ for a polynomial $\zeta(\lambda)$.*

- *G has a subgroup $F = \{(1+N)^x : x \in N^\zeta\}$ where discrete log (i.e., finding x) can be efficiently solved.*
- *G has a subgroup H that's isomorphic to $\mathbb{Z}^*_N$, and $G = F \times H$.*
- *Consider a random element $g \in G$ such that the Jacobi symbol of g mod N is 1. Then $\langle g \rangle$ contains F except with negligible probability. We write $g \leftarrow \mathsf{Samp}(N, \zeta)$ to mean sampling such elements g with Jacobi symbol 1.*

The following DDH assumption in Paillier groups is adapted from the formulation by [ADOS22], where the authors formulate a separate "small exponent" assumption stating it's secure to sample the secret exponents in DDH from a smaller, but still sufficiently large, range than the order of g. We directly state the small-exponent variant of DDH in Paillier groups here.

Definition 5.9 (DDH in Paillier Groups)**.** We say DDH holds in Paillier groups if the following holds for every polynomial $\zeta(\lambda)$:

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{DCR.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{DCR.Samp}(\mathsf{pp}),\ a, b \leftarrow [N]. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{DCR.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{DCR.Samp}(\mathsf{pp}),\ a, b, c \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

Lemma 5.10 (Distributed Discrete Log for Paillier Groups [RS21, OSY21])**.** *For a Pailler group $G = \mathbb{Z}^*_{N^{\zeta+1}}$, there exists an algorithm $\mathsf{DDLog}_{N,\zeta}$:*

- $\mathsf{DDLog}_{N,\zeta}(a \in G)$ *takes an element a and outputs a value $\alpha \in \mathbb{Z}_p$.*

The algorithm has the guarantee that for all $a \in G$ and $m \in [N^\zeta]$:

$$\Pr\left[\begin{array}{l} \mathsf{DDLog}_{N,\zeta}(a \cdot (1+N)^m) \\ = \mathsf{DDLog}_{N,\zeta}(a) + m \bmod N^\zeta \end{array} \right] = 1.$$

Remark 5.11. We note that the DDLog algorithm can be implemented in depth $O(\log \lambda)$, assuming $\log N < \mathrm{poly}(\lambda)$ and $\zeta < O(1)$, which suffices for our construction.

We also recall the ring LWE (RLWE) assumption over polynomial rings of the form $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, where $n(\lambda)$ is a power-of-2.

Definition 5.12 (RingLWE [LPR10])**.** We say the RLWE assumption holds with respect to the ring $\mathcal{R}(\lambda)$, a modulus $q(\lambda)$, error and secret distributions $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ if the following holds for every polynomial $m(\lambda)$:

$$\left\{ \begin{array}{l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}_1, \\ (\text{over } \mathcal{R}_\mathrm{q}) \end{array} \middle|\; \begin{array}{l} s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}_1, \leftarrow \mathcal{D}^m_{\mathsf{err}}, \\ \mathbf{a} \leftarrow \mathcal{R}^m_q \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b} \leftarrow \mathcal{R}^m_q \right\}_\lambda$$

5.2 Instantiation Under Groups

Our instantiations under prime-order groups and Pailler groups are analogous. Hence we focus on the prime-order groups in this section.

The starting point of our group instantiation is the DDH-based aHMAC scheme of [ILL25a]. In particular, it contains a pair of evaluation algorithms EvalKey, EvalTag that allows two parties P_G, P_E who jointly hold additive shares of the form $\langle \mathbf{s} \cdot x \rangle$, $\langle \mathbf{s} \cdot y \rangle$, to *locally* derive additive shares $\langle \mathbf{s}' \cdot g(x, y) \rangle$, for any Boolean gate g. [8] We will directly use these algorithms to implement our HGate protocol, and our main focus is to design compatible protocols HRead and HCompress for these encodings.

In more detail, in our group instantiation, the encoding of a wire value $[\![x]\!]$ for $x \in \{0, 1\}$ contains a pair of additive shares $\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle$ over $\mathbb{Z}_p$, and $\mathbf{s}^{\mathsf{id}} \in [p]^2$ are two secret exponents over a modulus p and with an associated id. Additionally, the value x is revealed to the evaluator.

$$[\![x]\!] = ([\![x]\!]_G, [\![x]\!]_E),\ \text{where } [\![x]\!]_G = (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_G, \mathsf{id}),\ \text{and } [\![x]\!]_E = (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_E, \mathsf{id}, x).$$

The $\mathsf{id} = (d, \mathsf{type})$ indicates the depth d of the current encoding and its type, i.e., whether it's an InWire (to depth d), or an OutWire. Our encoding of a hash value $[\![\mathbf{h}]\!]$ similarly contains a pair of additive shares $\langle \mathbf{s}^{\mathsf{id}} \otimes \mathbf{h} \rangle$ over $\mathbb{Z}_p$, an id with type = hash, but without requiring the evaluator to hold $\mathbf{h}$. [9]

Under this encoding, the Rekey steps are as follows, assuming access to public parameters pp that specify the modulus p.

[8] The DDH-based algorithms of [ILL25a] incurs a 1/poly correctness error probability. But we note the techniques from a follow-up [ILL25b] for removing this error can be adapted to our constructions.

[9] Note that the hash encoding format is still compatible with the described Rekey steps.

- $\mathsf{RekeySamp}(\mathsf{id})$ samples random share $\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_G \leftarrow [p]^2$, and sets $[\![x']\!]_G := (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_G, \mathsf{id})$.
- $\mathsf{RekeyGen}([\![x]\!]_G, [\![x']\!]_G)$ outputs the difference $\mathsf{rk} \leftarrow \langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_G - \langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_G$.
- $\mathsf{RekeyEval}([\![x]\!]_E, \mathsf{rk})$ computes $\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_E \leftarrow \langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_E + \mathsf{rk}$, and sets $[\![x']\!]_E := (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_E, \mathsf{id}, x)$.

In the following, we make extensive use of the fact that two parties can locally evaluate any linear functions on additive shares (over $\mathbb{Z}_p$). Examples include addition between shares, multiplication by any scalar $c \in \mathbb{Z}_p$, and bit-composition over shares of bits. When two parties want to derive any share $\langle a \rangle$, it suffices for them to derive $\langle \mathsf{Bits}(a) \rangle$. We will omit explicitely writing out bit-compositions in the following to avoid cluttering.

Instantiation of HRead **and** HCompress**.** We present the two sub-protocols HRead, HCompress in Figure 1, and 2. We note that the sub-protocols only run the local algorithms $\mathsf{HRead}_G, \mathsf{HRead}_E, \mathsf{HCompress}_G, \mathsf{HCompress}_E$, (Construction 5.18) and don't involve communication between the parties. The sub-protocols assume the parties P_G, P_E hold public data pd produced in the overall protocol by Enc. (See Equation 5.4.)

Protocol HRead

The protocol runs between a garbler P_G and an evaluator P_E. It uses the following ingradients:

- The algorithms $\mathsf{HRead}_G, \mathsf{HRead}_E$, from Construction 5.18.

Inputs: P_G, P_E hold public data pd, index i, and jointly encodings $[\![\mathbf{h}]\!]$ (with $\mathsf{id} = (d, \mathsf{hash})$) of a hash and P_E additionally holds encodings $[\![\mathbf{x}]\!]_E$ (without requirements on their ids) of the hashed values.
Outputs: P_G, P_E jointly hold encodings $[\![x_i]\!]$ (with $\mathsf{id}' = (d+1, \mathsf{InWire})$) of the i-th hashed value.

- P_G reads $\mathsf{HR.pd}^{(d)}$ from pd, $\langle \mathbf{s}^{\mathsf{id}} \otimes \mathbf{h} \rangle_G$ from $[\![\mathbf{h}]\!]_G$, and runs

$$\langle \mathbf{s}^{\mathsf{id}'} \cdot x_i \rangle_G \leftarrow \mathsf{HRead}_G^{\mathsf{Hash.pd}^{(d)}}(\langle \mathbf{s}^{\mathsf{id}} \otimes \mathbf{h} \rangle_G, i).$$

  It outputs $[\![x_i]\!]_G := (\langle \mathbf{s}^{\mathsf{id}'} \cdot x_i \rangle_G, \mathsf{id}')$.

- P_E reads $\mathsf{HR.pd}^{(d)}$ from pd, $\langle \mathbf{s}^{\mathsf{id}} \otimes \mathbf{h} \rangle_E$ from $[\![\mathbf{h}]\!]_E$, $\mathbf{x}$ from $[\![\mathbf{x}]\!]_E$, and runs

$$\langle \mathbf{s}^{\mathsf{id}'} \cdot x_i \rangle_E \leftarrow \mathsf{HRead}_E^{\mathsf{Hash.pd}^{(d)}}(\langle \mathbf{s}^{\mathsf{id}} \otimes \mathbf{h} \rangle_E, i, \mathbf{x}).$$

  It outputs $[\![x_i]\!]_E := (\langle \mathbf{s}^{\mathsf{id}'} \cdot x_i \rangle_E, \mathsf{id}', \mathbf{x}[i])$.

Figure 1: The HRead sub-protocol.

Below, we first present our core construction of a pair of fixed-factor hashing algorithms in Construction 5.15, which uses a SecSwitch algorithm from Construction 5.13 as a building block. Then, we describe wrapper algorithms $\mathsf{HRead}_G, \mathsf{HRead}_E, \mathsf{HCompress}_G, \mathsf{HCompress}_E$ that call fixed-factor hashing in Construction 5.18.

Construction 5.13 (Secret Switching for aHMAC Encodings)**.** The construction is with respect to the following public parameters pp:

- A bound $B < \mathsf{poly}(\lambda)$ on encoded values and $0 < \delta < 1$ on correctness error probability.
- A DDH group instance G with a prime order $p > \lambda^{\omega(1)}$ and a generator $g \in G$.

It uses the DDLog algorithm from Lemma 5.5 instantiated with a suitable $\mathsf{PRF} : \{0,1\}^\lambda \times G \to \{0,1\}^\lambda$ to achieve error probability $< \delta$.

Protocol HCompress

The protocol runs between a garbler P_G and an evaluator P_E. It uses the following ingradients:

- The algorithms $\mathsf{HCompress}_G, \mathsf{HCompress}_E$ from Construction 5.18.

Inputs: P_G, P_E hold public data pd and jointly encodings $[\![\mathbf{x}]\!]$ (with $\mathsf{id} = (d, \mathsf{OutWire})$).
Outputs: P_G, P_E jointly hold the encodings $[\![\mathbf{h}]\!]$ (with $\mathsf{id}' = (d, \mathsf{hash})$) of the hash.

- P_G reads $\mathsf{HW.pd}^{(d)}$ from pd, $\langle \mathbf{s}^{\mathsf{id}} \otimes \mathbf{x} \rangle$ from $[\![x'_i]\!]_G$ for $i \in S$, and runs

$$\langle \mathbf{s}^{\mathsf{id}'} \otimes \mathbf{h} \rangle_G \leftarrow \mathsf{HCompress}_G^{\mathsf{HW.pd}^{(d)}}(\langle \mathbf{s}^{\mathsf{id}} \otimes \mathbf{x} \rangle_G).$$

  It outputs $[\![\mathbf{h}]\!]_G := (\langle \mathbf{s}^{\mathsf{id}'} \otimes \mathbf{h} \rangle_G, \mathsf{id}')$.

- P_E analogously runs $\mathsf{HCompress}_E$ as follows:

$$\langle \mathbf{s}^{\mathsf{id}'} \otimes \mathbf{h} \rangle_E \leftarrow \mathsf{HCompress}_E^{\mathsf{HW.pd}^{(d)}}(\langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x} \rangle_E, \mathbf{x}).$$

  where $\mathbf{x}$ is read from $[\![\mathbf{x}]\!]_E$. It outputs $[\![\mathbf{h}]\!]_E := (\langle \mathbf{s}^{\mathsf{id}'} \otimes \mathbf{h} \rangle_E, \mathsf{id}')$.

Figure 2: The HCompress sub-protocol.

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s} \in [p]^2, \mathbf{s}' \in \{0,1\}^\ell)$**:** Compute the secret-switching public data as

$$\mathbf{r} \leftarrow [p]^\ell,\ \mathsf{pd} \quad := (\mathbf{c}_0 = g^{\mathbf{r}}, \mathbf{c}_1 = g^{\mathbf{s}[0]\cdot\mathbf{r}+\mathbf{s}'})$$

Additionally, instantiate $\mathsf{DDLog}_{G,g}$ with a (public) random PRF key $\leftarrow \{0,1\}^\lambda$ which is also included in pd. Since invokations of PRF happens implicitly within invocations of $\mathsf{DDLog}_{G,g}$, we omit it from the following descriptions and similarly from other algorithms in this section.

$\langle \mathbf{s}' \cdot x \rangle_{G/E} \leftarrow \mathsf{SecSwitch}^{\mathsf{pd}}(\langle \mathbf{s} \cdot x \rangle_{G/E}, \langle x \rangle_{G/E})$**:** Read $(\mathbf{c}_0, \mathbf{c}_1)$ from pd and compute the following

$$\mathbf{v}_{G/E} \leftarrow (\mathbf{c}_1)^{\langle x \rangle_{G/E}} / (\mathbf{c}_0)^{\langle \mathbf{s}[0]\cdot x \rangle_{G/E}},$$
$$\langle \mathbf{s}' \cdot x \rangle_{G/E} \leftarrow \mathsf{DDLog}_{G,g}(\mathbf{v}_{G/E}) \bmod p.$$

We show the following correctness lemma.

Lemma 5.14 (Correctness of Secret Switching)**.** *For every polynomial length bound $\ell(\lambda)$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$, public parameters* pp *as specified, encoded value $x \in [B]$, secrets $\mathbf{s} \in [p]^2, \mathbf{s}' \in [p]^\ell$, additve shares $\langle \mathbf{s} \cdot x \rangle$, $\langle x \rangle$, the following holds:*

$$\Pr\left[\langle \mathbf{s}' \cdot x \rangle_E = \langle \mathbf{s}' \cdot x \rangle_G + \mathbf{s}' \cdot x \;\middle|\; \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \cdot x \rangle_G \leftarrow \mathsf{SecSwitch}(\langle \mathbf{s} \cdot x \rangle_G, \langle x \rangle_G), \\ \langle \mathbf{s}' \cdot x \rangle_E \leftarrow \mathsf{SecSwitch}(\langle \mathbf{s} \cdot x \rangle_E, \langle x \rangle_E) \end{array} \right] > 1 - \mathsf{negl}(\lambda) - \delta(\lambda).$$

Proof. We first check that the intermediate values $\mathbf{v}_E, \mathbf{v}_G$ during SecSwitch satsify $\mathbf{v}_E = \mathbf{v}_G \cdot g^{\mathbf{s}'\cdot x}$.

$$\begin{aligned} \mathbf{v}_E &= \mathbf{c}_1^{\langle x \rangle_E} / \mathbf{c}_0^{\langle \mathbf{s}[0]\cdot x \rangle_E} \\ (\text{By Construction}) &= \mathbf{c}_1^{\langle x \rangle_G + x} / \mathbf{c}_0^{\langle \mathbf{s}[0]\cdot x \rangle_G + \mathbf{s}[0]\cdot x} = \mathbf{v}_G \cdot \mathbf{c}_1^x / \mathbf{c}_0^{\mathbf{s}[0]\cdot x} \\ (\text{By Setup}) &= \mathbf{v}_G \cdot g^{(\mathbf{s}[0]\cdot\mathbf{r}+\mathbf{s}')\cdot x} / g^{\mathbf{r}\cdot\mathbf{s}[0]\cdot x} = \mathbf{v}_G \cdot g^{(\mathbf{s}'\cdot x)}. \end{aligned}$$

Then by the correctness of DDLog (Lemma 5.5), we have $\mathsf{DDLog}(\mathbf{v}_E) = \mathsf{DDLog}(\mathbf{v}_G) + \mathbf{s}' \cdot x \bmod p$, as desired. □

Construction 5.15 (Fixed Factor Hashing for aHMAC Encodings under DDH)**.** The construction is with respect to the same public parameters $\mathsf{pp} = (B, \delta, G, p, g)$ as specified in Construction 5.13, and additionally

- A hash function $\mathsf{Hash} : \mathbb{Z}^W \to \mathbb{Z}^{\lceil \log p \rceil}$ with compression factor W, *implicitely* defined by W (the exponents of) group elements g^{b_i} for $i \in [W]$;

$$\mathsf{Hash}(\mathbf{x} \in \mathbb{Z}_B^W) := \sum_i \mathsf{Bits}(b_i) \cdot \mathbf{x}[i] \text{ over } \mathbb{Z} \tag{5.1}$$

It uses as ingradients the secret switching algorithms (Construction 5.13, denoted with prefix SS) and the DDLog algorithm from Lemma 5.5 instantiated with a suitable $\mathsf{PRF} : \{0,1\}^\lambda \times G \to \{0,1\}^\lambda$ to achieve error probability $< \delta$.

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s}, \mathbf{s}' \in [p]^2)$**:** Read $\{g^{b_i}\}_{i\in[W]}$ from pp, and generate ciphertexts $\{\mathsf{ct}_{i,j,b}\}_{i,j\in[W],b\in\{0,1\}}$ used for extracting hashed values:

$$\mathbf{a}_j \leftarrow [p]^{\lceil \log p \rceil},$$
$$\mathsf{ct}_{i,j,b} \leftarrow (g^{b_i})^{\mathbf{s}[b]\cdot \mathbf{a}_j} / g^{\mathbf{u}_j[i]\cdot \mathsf{Bits}(\mathbf{s}'[b])} \in G$$

where $\mathbf{u}_j \in \{0,1\}^W$ denote the j-th unit vector. Next, generate public data $\{\mathsf{pd}_{i,b}\}_{i\in[W],b\in\{0,1\}}$ for computing the hash over shares: [10]

$$\mathsf{pd}_{i,b} \leftarrow \mathsf{SS.Setup}(1^\lambda, \mathbf{s}, \mathsf{Bits}(\mathbf{s}'[b]) \otimes \mathsf{Bits}(b_i)).$$

Finally, output

$$\mathsf{pd} := (\mathsf{pp}, \{\mathsf{ct}_{i,j,b}\}, \{g^{\mathbf{a}_j}\}, \{\mathsf{pd}_i\}).$$

$\langle \mathbf{s}' \otimes \mathbf{h} \rangle_{G/E} \leftarrow \mathsf{Compress}^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{x} \rangle_{G/E}, \langle \mathbf{x} \rangle_{G/E})$**:** Read $\{\mathsf{pd}_{i,b}\}_{i\in[W],b\in\{0,1\}}$ from pd, and compute

$$\langle \mathbf{s}' \otimes \mathbf{h} \rangle_{G/E} \leftarrow \sum_i \langle \mathbf{s}' \otimes \mathsf{Bits}(b_i) \cdot \mathbf{x}[i] \rangle_{G/E}, \text{ where for } i \in [W], b \in \{0,1\}:$$
$$\langle \mathsf{Bits}(\mathbf{s}'[b]) \otimes \mathsf{Bits}(b_i) \cdot \mathbf{x}[i] \rangle_{G/E} \leftarrow \mathsf{SS.SecSwitch}^{\mathsf{pd}_{i,b}}(\langle \mathbf{s} \cdot \mathbf{x}[i] \rangle_{G/E}, \langle \mathbf{x}[i] \rangle_{G/E}).$$

$\langle \mathbf{s}' \cdot x_j \rangle_{G/E} \leftarrow \mathsf{Extract}^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{h} \rangle_{G/E}, j, \langle \mathbf{x} \rangle_{G/E})$**:** First read $\{\mathsf{ct}_{i,j,b}\}_{i,j\in[W],b\in\{0,1\}}$ from pd, and compute

$$\mathsf{diff}_{G/E,j,b} \leftarrow \prod_i (\mathsf{ct}_{i,j,b})^{\langle \mathbf{x}[i] \rangle_{G/E}}.$$

Next read $g^{\mathbf{a}_j}$ and $\alpha, \beta \in [p]^{\lceil \log p \rceil}$ from pd, and compute

$$\langle \mathbf{s} \cdot h \rangle_{G/E} \leftarrow \sum_{k\in[\lceil \log p \rceil]} 2^k \cdot \langle \mathbf{s} \cdot \mathbf{h}[k] \rangle_{G/E},$$
$$\mathbf{v}_{G/E,b} \leftarrow (g^{\mathbf{a}_j})^{\langle \mathbf{s}[b]\cdot h \rangle_{G/E}} / \mathsf{diff}_{G/E,j,b},$$
$$\langle \mathsf{Bits}(\mathbf{s}'[b]) \cdot x_j \rangle_{G/E} \leftarrow \mathsf{DDLog}_{G,g}(\mathbf{v}_{G/E,b}) \bmod p.$$

We show the following correctness lemma.

Lemma 5.16 (Correctness of Fixed Factor Hashing)**.** *There exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$*, public parameters* pp *as specified, encoded vectors* $\mathbf{x} \in [B]^W$*, secrets* $\mathbf{s}, \mathbf{s}' \in [p]^2$*, additve shares* $\langle \mathbf{s} \otimes \mathbf{x} \rangle$*,* $\langle \mathbf{x} \rangle$*,* $\langle \mathbf{s} \otimes \mathsf{Hash}(\mathbf{x}) \rangle$*, and index* $j \in [W]$*, the following holds:*

$$\Pr\left[\langle \mathbf{s}' \otimes \mathbf{h} \rangle_E = \langle \mathbf{s}' \otimes \mathbf{h} \rangle_G + \mathbf{s}' \otimes \mathsf{Hash}(\mathbf{x}) \,\middle|\, \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \otimes \mathbf{h} \rangle_G \leftarrow \mathsf{Compress}(\langle \mathbf{s} \otimes \mathbf{x} \rangle_G, \langle \mathbf{x} \rangle_G), \\ \langle \mathbf{s}' \otimes \mathbf{h} \rangle_E \leftarrow \mathsf{Compress}(\langle \mathbf{s} \otimes \mathbf{x} \rangle_E, \langle \mathbf{x} \rangle_E) \end{array} \right] > 1 - \mathsf{negl}(\lambda) - \delta(\lambda).$$

$$\Pr\left[\langle \mathbf{s}' \cdot x_j \rangle_E = \langle \mathbf{s}' \cdot x_j \rangle_G + \mathbf{s}' \cdot \mathbf{x}[j] \,\middle|\, \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \cdot x_j \rangle_G \leftarrow \mathsf{Extract}_G(\langle \mathbf{s} \otimes \mathsf{Hash}(\mathbf{x}) \rangle_G, j, \langle \mathbf{x} \rangle_G), \\ \langle \mathbf{s}' \cdot x_j \rangle_E \leftarrow \mathsf{Extract}_E(\langle \mathbf{s} \otimes \mathsf{Hash}(\mathbf{x}) \rangle_E, j, \langle \mathbf{x} \rangle_E), \end{array} \right] > 1 - \mathsf{negl}(\lambda) - \delta(\lambda).$$

[10] We abuse notation here, writing $\mathsf{Bits}(b_i) \otimes \mathsf{Bits}(\mathbf{s}[b])$ to mean the flattened vector as the input to SS.Setup.

Proof. The correctness of Compress follows directly from that of SecSwitch and the definition of Hash. To verify the correctness of Extract, it suffices to check the intermediate values $\mathbf{v}_{G,b}$ and $\mathbf{v}_{E,b}$ satisfy $\mathbf{v}_{E,b} = \mathbf{v}_{G,b} \cdot g^{\mathsf{Bits}(\mathbf{s}'[b])\cdot x_j}$. (Then, correctness of Extract follows from that of DDLog.)

$$\begin{aligned}
\mathbf{v}_{E,b} &= (g^{\mathbf{a}_j})^{\langle \mathbf{s}[b]\cdot h\rangle_E} / \mathrm{diff}_{E,j,b} \\
\text{(By Construction)} &= (g^{\mathbf{a}_j})^{\langle \mathbf{s}[b]\cdot h\rangle_G + \mathbf{s}[b]\cdot h} / \prod_i \mathrm{ct}_{i,j,b}^{\langle \mathbf{x}[i]\rangle_G + \mathbf{x}[i]} \\
&= \mathbf{v}_{G,b} \cdot (g^{\mathbf{a}_j})^{\mathbf{s}[b]\cdot h} / \prod_i \mathrm{ct}_{i,j,b}^{\mathbf{x}[i]} \\
\text{(By Setup)} &= \mathbf{v}_{G,b} \cdot (g^{\mathbf{a}_j})^{\mathbf{s}[b]\cdot h} / \left(\prod_i g^{\mathbf{s}[b]\cdot b_i \cdot \mathbf{a}_j \cdot \mathbf{x}[i]}\right) \cdot \left(\prod_i g^{\mathbf{u}_j[i]\cdot \mathsf{Bits}(\mathbf{s}'[b])\cdot \mathbf{x}[i]}\right) \\
&= \mathbf{v}_{G,b} \cdot \left(g^{\mathbf{a}_j \cdot \mathbf{s}[b]}\right)^{h - \sum_i b_i \cdot x_i} \cdot g^{\mathsf{Bits}(\mathbf{s}'[b])\cdot x_j}.
\end{aligned}$$

It suffices to check that $h = \sum_i b_i x_i$:

$$\begin{aligned}
h &:= \sum_{k \in [\lceil \log p \rceil]} 2^k \cdot \mathbf{h}[k] \\
\text{(By Definition)} &= \sum_{k \in [\lceil \log p \rceil]} \sum_i 2^k \cdot \mathsf{Bits}(b_i)[k] \cdot x_i = \sum_i b_i \cdot x_i.
\end{aligned}$$

□

For our security proof later, we note here that the combined public data pd produced by Setup from fixed factor hashing (denoted with prefix FH) and secret switching (denoted with prefix SS) under a random secret $\mathbf{s} \in [p]^2$ are computationally indistinguishable from random. The following lemma follows directly from the DDH assumption (Definition 5.4), and hence we omit its proof.

Lemma 5.17 (Security of Fixed Factor Hashing)**.** *Under the DDH assumption, every efficient adversary $\mathcal{A}$ has negligible advantage in distinguishing the following experiments $\mathsf{Exp}^b_{\mathcal{A}}(1^\lambda)$, for $b \in \{0,1\}$*

- *Sample public parameters* pp*, and sends them to $\mathcal{A}$, who returns exponents $\mathbf{s}' \in [p]^2$.*
- *When $b = 0$, sample random exponents $\mathbf{s} \leftarrow [p]^2$ and send* pd *to $\mathcal{A}$, computed as follows*

$$\mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}').$$

- *When $b = 1$, sample* pd *with random group elements from G and send them to $\mathcal{A}$.*

Construction 5.18 (Hashing for aHMAC Encodings)**.** The construction is with respect to the following public parameters pp:

- A bound γ on the number of parallel instances of fixed factor hashing, and $0 < \delta < 1$ on the correctness error probability.
- A DDH group instance G with a prime order $p > \lambda^{\omega(1)}$ and a generator $g \in G$.
- A hash function $\mathsf{Hash} : \mathbb{Z}^W \to \mathbb{Z}^{\lceil \log p \rceil}$ defined as in Equation 5.1.

It uses the fixed factor hashing algorithms (Construction 5.15, denoted with prefix FH) as ingradients, with respect to the above G, p, g, Hash and the following additional parameters B, δ'.

- A bound $B = W$ on encoded values;
- An error bound $\delta' < \delta/\gamma$ (s.t. the error probability over all invocations of the ingradient algorithms is $< \delta$).

For a string $\mathbf{x} \in \{0,1\}^\ell$ of length $\ell < W \cdot \gamma$, we define its hash $h_{\mathbf{x}}$ to be the γ hashed values of W chunks of $\mathbf{x}$ by Hash (Equation 5.1). For an index $i < W \cdot \gamma$, we define $\mathsf{Chunk}_i \in [\gamma]$ to be the (index of the) chunk corresponding to $\mathbf{x}[i]$ during hashing.

$$\mathsf{Chunk}_i = \lfloor i/W \rfloor. \tag{5.2}$$

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s}, \mathbf{s}' \in [p]^2)$**:** Prepare public data for an instance of fixed factor hashing:

$$\mathsf{FH.pd} \leftarrow \mathsf{FH.Setup}(1^\lambda, \mathbf{s}, \mathbf{s}').$$

Output $\mathsf{pd} := \mathsf{FH.pd}$.

$\langle \mathbf{s}' \cdot x_i \rangle_G \leftarrow \mathsf{HRead}_G^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{h} \rangle_G, i)$**:** Assumeing $i < W \cdot \gamma$, let $j := \mathsf{Chunk}_i$ as defined in Equation 5.7 and $i' := i \bmod W$. Read FH.pd and run the following:

$$\langle \mathbf{s} \otimes \mathbf{h}_j \rangle_G \leftarrow \langle \mathbf{s} \otimes \mathbf{h}[j \cdot \lceil \log p \rceil : (j+1) \cdot \lceil \log p \rceil] \rangle_G,$$
$$\langle \mathbf{s}' \cdot x_i \rangle_G \leftarrow \mathsf{FH.Extract}^{\mathsf{FH.pd}}(\langle \mathbf{s} \otimes \mathbf{h}_j \rangle_G, i', \langle \mathbf{x}_j \rangle_G) \text{ where } \langle \mathbf{x}_j \rangle_G \leftarrow \mathbf{0}.$$

$\langle \mathbf{s}' \cdot x_i \rangle_E \leftarrow \mathsf{HRead}_E^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{h} \rangle_E, i, \mathbf{x})$**:** Analogous to HRead_G, except running FH.Extract with $\langle \mathbf{x}_j \rangle_E$ defined as:

$$\mathbf{x}_j := \mathbf{x}[j \cdot W : (j+1) \cdot W], \ \langle \mathbf{x}_j \rangle_E \leftarrow \mathbf{x}_j.$$

$\langle \mathbf{s}' \otimes \mathbf{h} \rangle_G \leftarrow \mathsf{HCompress}_G^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{x} \rangle_G)$**:** Read FH.pd from pd, and run FH.Compress on chunks of $\langle \mathbf{s} \otimes \mathbf{x} \rangle_G$ and $\langle \mathbf{x} \rangle_G \leftarrow \mathbf{0}$. For $j \in [\gamma]$:

$$\langle \mathbf{s} \otimes \mathbf{x}_j \rangle_G \leftarrow \langle \mathbf{s} \otimes \mathbf{x}[j \cdot W : (j+1) \cdot W] \rangle_G,$$
$$\langle \mathbf{x}_j \rangle_G \leftarrow \langle \mathbf{x}[j \cdot W : (j+1) \cdot W] \rangle_G,$$
$$\langle \mathbf{s}' \otimes \mathbf{h}_j \rangle_G \leftarrow \mathsf{FH.Compress}^{\mathsf{FH.pd}^{(1)}}(\langle \mathbf{s} \otimes \mathbf{x}_j \rangle_G, \langle \mathbf{x}_j \rangle_G),$$

Finally, concatenate the obatined hashs as $\langle \mathbf{s}' \otimes \mathbf{h} \rangle_G$ and output it.

$\langle \mathbf{s}' \otimes \mathbf{h} \rangle_E \leftarrow \mathsf{HCompress}_E^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{x} \rangle_E, \mathbf{x})$**:** Analogous to $\mathsf{HCompress}_G$, except running FH.Compress with chunks of $\langle \mathbf{x} \rangle_E \leftarrow \mathbf{x}$.

Remark 5.19. As presented, a hash computed by HCompress has a fixed size $\gamma \cdot \mathsf{poly}(\lambda)$. In our application to garbling schemes, we optimize to only run $\lceil |\mathbf{x}|/W \rceil$, instead of γ, instances of the fixed factor hash. The resuling hash has size $\lceil |S|/W \rceil \cdot \mathsf{poly}(\lambda)$.

We note the following correctness lemma, which follows from those of Compress, Extract.

Lemma 5.20 (Correctness of Hashing)**.** *There exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$, public parameters* pp *as specified, encoded vectors $\mathbf{x} \in \{0,1\}^{W \cdot \gamma}$, index $j \in [W \cdot \gamma]$, secrets $\mathbf{s}, \mathbf{s}' \in [p]^2$, additve shares $\langle \mathbf{s} \otimes \mathbf{x} \rangle$, $\langle \mathbf{s} \otimes \mathbf{h} \rangle$, where $\mathbf{h}$ is hash of $\mathbf{x}$,*

$$\Pr\left[\langle \mathbf{s}' \cdot x_j \rangle_E = \langle \mathbf{s}' \cdot x_j \rangle_G + \mathbf{s}' \cdot \mathbf{x}[j] \;\middle|\; \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \cdot x_j \rangle_G \leftarrow \mathsf{HRead}_G(\langle \mathbf{s} \otimes \mathbf{h} \rangle_G, j), \\ \langle \mathbf{s}' \cdot x_j \rangle_E \leftarrow \mathsf{HRead}_E(\langle \mathbf{s} \otimes \mathbf{h} \rangle_E, j, \mathbf{x}), \end{array}\right] > 1 - \mathsf{negl}(\lambda) - \delta(\lambda)$$

$$\Pr\left[\langle \mathbf{s}' \otimes \mathbf{h} \rangle_E = \langle \mathbf{s}' \otimes \mathbf{h} \rangle_G + \mathbf{s}' \otimes \mathbf{h}, \;\middle|\; \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \otimes \mathbf{h} \rangle_G \leftarrow \mathsf{HCompress}_G(\langle \mathbf{s} \otimes \mathbf{x} \rangle_G), \\ \langle \mathbf{s}' \otimes \mathbf{h} \rangle_E \leftarrow \mathsf{HCompress}_E(\langle \mathbf{s} \otimes \mathbf{x} \rangle_E, \mathbf{x}), \end{array}\right] > 1 - \mathsf{negl}(\lambda) - \delta(\lambda).$$

For our security proof later, we note the following lemma, which follows from Lemma 5.17.

Lemma 5.21 (Security of Hashing)**.** *Under the DDH assumption, every efficient adversary $\mathcal{A}$ has negligible advantage in distinguishing the experiments $\mathsf{Exp}^b_{\mathcal{A}}(1^\lambda)$, for $b \in \{0,1\}$, defined analogously to Lemma 5.17 with respect to the* Setup *algorithm in Construction 5.18.*

Instantiation of HGate**.** We set up our encodings for wire values to be directly compatible with the aHMAC evaluation constructions from [ILL25a], which are used for the HGate protocol (Figure 3). We note that HGate only runs local algorithms EvalKey, EvalTag (Construction 5.24) described below, and doesn't involve communication between the parties. Similarly to HRead and HCompress, the sub-protocol assume the parties P_G, P_E hold public data pd produced in the overall protocol by Enc. (See Equation 5.4.)

Protocol HGate

The protocol runs between a garbler P_G and an evaluator P_E. It uses the following ingradients:

- The algorithms EvalKey, EvalTag from Construction 5.24.

Inputs: P_G, P_E hold public data pd, an operator op, and jointly encodings of $[\![x]\!]$, $[\![y]\!]$, (with id = (d, InWire)).
Outputs: P_G, P_E jointly hold encodings $[\![z]\!]$ (with $\mathsf{id}' = (d, \mathsf{OutWire})$), where z should equal $\mathsf{op}(x, y)$.

- P_G reads $\mathsf{GE.pd}^{(d)}$ from pd, $\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_G$ $\langle \mathbf{s}^{\mathsf{id}} \cdot y \rangle_G$ from $[\![x]\!]_G, [\![x]\!]_G$, and runs

$$\langle \mathbf{s}^{\mathsf{id}'} \cdot z \rangle_G \leftarrow \mathsf{EvalKey}^{\mathsf{GE.pd}^{(d)}}(\mathsf{op}, \langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_G, \langle \mathbf{s}^{\mathsf{id}} \cdot y \rangle_G).$$

  It outputs $[\![z]\!]_G := (\langle \mathbf{s}^{\mathsf{id}'} \cdot z \rangle_G, \mathsf{id}')$.

- P_E proceeds analogously to P_G by running EvalTag, and outputs $[\![z]\!]_E := (\langle \mathbf{s}^{\mathsf{id}'} \cdot z \rangle_E, \mathsf{id}', \mathsf{op}(x, y))$.

Figure 3: The HGate sub-protocol.

We recall neccessary preliminaries from [ILL25a] below (modified slightly for a more modular presentation). When possible, we omit construction details that are not relevant for security proofs and only describe the interfaces.

Construction 5.22 (aHMAC Multiplication under DDH [ILL25a])**.** The construction is with respect to the same public parameters $\mathsf{pp} = (B, \delta, G, p, g)$ as specified in Construction 5.13.

It uses as ingredients the DDLog algorithm from Lemma 5.5 instantiated with a suitable $\mathsf{PRF} : \{0,1\}^\lambda \times G \to \{0,1\}^\lambda$ to achieve error probability $< \delta$.

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, s, s', s'' \in [p])$**:** takes three secrets $s, s', s'' \in [p]$, and computes pd as follows.

$$\mathbf{r} \leftarrow [p]^{\lceil \log p \rceil},\ \mathsf{pd} := (\mathsf{pp}, g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s's}, g^{\mathbf{r}ss'+\mathsf{Bits}(s'')}).$$

$\langle s'' \cdot z \rangle_G \leftarrow \mathsf{MultKey}^{\mathsf{pd}}(\langle s \cdot x \rangle_G, \langle s' \cdot y \rangle_G)$**:** takes as inputs two integers representing the 0-shares of $\langle s \cdot x \rangle_G, \langle s' \cdot y \rangle_E \in \mathbb{Z}$ for some secrets $s, s' \in [p]$. It outputs the 0-share of another share $\langle s'' \cdot z \rangle_G$.

$\langle s'' \cdot z \rangle_E \leftarrow \mathsf{MultTag}^{\mathsf{pd}}(\langle s \cdot x \rangle_E, \langle s' \cdot y \rangle_E, x, y)$**:** takes as input the 1-shares of $\langle s \cdot x \rangle_E, \langle s' \cdot y \rangle_E$ for some secret s, s', and the shared values $x, y \in [B]$. It outputs the 1-share of another share $\langle s'' \cdot z \rangle_E$.

Lemma 5.23 (Correctness of aHMAC Multiplication [ILL25a])**.** *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, public parameters* pp *as specified, secret elements* $s, s', s'' \in [p]$*, inputs* $x, y \in [B]$ *such that* $xy < B$*, and additive shares* $\langle sx \rangle_G, \langle sx \rangle_E, \langle s'y \rangle_G, \langle s'y \rangle_E$*, the following holds:*

$$\Pr\left[\begin{array}{l} w_E = w_G + s'' \cdot xy \\ (\text{over } \mathbb{Z}_p) \end{array} \middle| \begin{array}{l} \mathsf{pd} \leftarrow \mathsf{Setup}(\mathsf{pp}, s, s', s'') \\ w_G = \mathsf{MultKey}(\langle sx \rangle_G, \langle s'y \rangle_G) \\ w_E = \mathsf{MultTag}(\langle sx \rangle_E, \langle s'y \rangle_E, x, y) \end{array}\right] > 1 - \mathsf{negl}(\lambda) - \delta(\lambda).$$

We next recall from [ILL25a] the leveled gate evalaution algorithms EvalKey, EvalTag for aHMAC.

Construction 5.24 (aHMAC (leveled) Gate Evaluation [ILL25a])**.** The construction is with respect to the same public parameters $\mathsf{pp} = (\delta, G, p, g)$ as specified in Construction 5.22.

It uses Construction 5.22 (with prefix Mt) and Construction 5.13 (with prefix SS) as ingradients w.r.t. to G, p, g from above and $B = 2$, $\delta' = \delta/2$.

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s}, \mathbf{s}' \in [p]^2)$**:** prepare two instances of aHMAC multiplication, and an public data for secret switching from $\mathbf{s}$ to $\mathbf{s}'$.

$$\mathsf{Mt.pd}_0 \leftarrow \mathsf{Mt.Setup}(1^\lambda, \mathbf{s}[0], \mathbf{s}[1], \mathbf{s}'[0]),\ \mathsf{Mt.pd}_1 \leftarrow \mathsf{Mt.Setup}(1^\lambda, \mathbf{s}[0], \mathbf{s}[1], \mathbf{s}'[1]),$$
$$\mathsf{pd} := (\mathsf{Mt.pd}_0, \mathsf{Mt.pd}_1, \mathsf{SS.pd}),\ \mathsf{SS.pd} \leftarrow \mathsf{SS.Setup}(1^\lambda, \mathbf{s}, \mathbf{s}').$$

$\langle \mathbf{s}' \cdot z \rangle_G \leftarrow \mathsf{EvalKey}^{\mathsf{pd}}(g, \langle \mathbf{s} \cdot x \rangle_G, \langle \mathbf{s} \cdot y \rangle_G)$**:** We show the case for $g = \mathsf{XOR}$. (The case for $g = \mathsf{AND}$ or any binary gate is analogous.) First read SS.pd from pd, and compute $\langle \mathbf{s}' \cdot (x+y) \rangle_G$:

$$\langle \mathbf{s}' \cdot (x+y) \rangle_G \leftarrow \mathsf{SecSwitch}^{\mathsf{SS.pd}}(\langle \mathbf{s} \cdot (x+y) \rangle_G, \langle x+y \rangle_G),$$
$$\text{where } \langle \mathbf{s} \cdot (x+y) \rangle_G, \leftarrow \langle \mathbf{s} \cdot x \rangle_G + \langle \mathbf{s} \cdot y \rangle_G,\ \langle x+y \rangle_G \leftarrow 0.$$

Then read $\mathsf{Mt.pd}_0, \mathsf{Mt.pd}_1$ from pd, and compute the final output $\langle \mathbf{s}' \cdot z \rangle_G$ as follows:

$$\langle \mathbf{s}'[b] \cdot xy \rangle_G \leftarrow \mathsf{MultKey}^{\mathsf{Mt.pd}_b}(\langle \mathbf{s}[0] \cdot x \rangle_G, \langle \mathbf{s}[1] \cdot y \rangle_G),\ \text{for } b \in \{0,1\},$$
$$\langle \mathbf{s}' \cdot z \rangle_G \leftarrow 2 \cdot \langle \mathbf{s}' \cdot (x+y) \rangle_G - \langle \mathbf{s}' \cdot xy \rangle_G.$$

$\langle \mathbf{s}' \cdot z \rangle_E \leftarrow \mathsf{EvalTag}^{\mathsf{pd}}(g, \langle \mathbf{s} \cdot x \rangle_E, \langle \mathbf{s} \cdot y \rangle_E, x, y)$**:** Analogous to EvalKey, except when running SecSwitch, set $\langle x+y \rangle_E \leftarrow x+y$, and run MultTag instead of MultKey in the second step.

We note the following straightforward correctness and security lemmas and omit their proofs.

Lemma 5.25 (Correctness of aHMAC Gate Evaluation)**.** *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, public parameters* pp *as specified, secret elements* $\mathbf{s}, \mathbf{s}' \in [p]^2$*, inputs* $x, y \in \{0,1\}$*, binary gates* $g : \{0,1\}^2 \to \{0,1\}$ *and additive shares* $\langle \mathbf{s} \cdot x \rangle_G, \langle \mathbf{s} \cdot x \rangle_E, \langle \mathbf{s} \cdot y \rangle_G, \langle \mathbf{s} \cdot y \rangle_E$*, the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_E = \mathbf{w}_G + \mathbf{s}' \cdot g(x,y) & \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}') \\ (\text{over } \mathbb{Z}_p) & \mathbf{w}_G = \mathsf{EvalKey}(g, \langle \mathbf{s} \cdot x \rangle_G, \langle \mathbf{s} \cdot y \rangle_G) \\ & \mathbf{w}_E = \mathsf{EvalTag}(g, \langle \mathbf{s} \cdot x \rangle_E, \langle \mathbf{s} \cdot y \rangle_E, x, y) \end{array}\right] > 1 - \mathsf{negl}(\lambda).$$

Lemma 5.26 (Security of aHMAC Gate Evaluation)**.** *Under the DDH assumption, every efficient adversary* $\mathcal{A}$ *has negligible advantage in distinguishing the experiments* $\mathsf{Exp}^b_{\mathcal{A}}(1^\lambda)$*, for* $b \in \{0,1\}$*, defined analogously to Lemma 5.17 with respect to the* Setup *algorithm in Construction 5.24.*

Instantiation of Setup **and** Enc**.** We now describe the $\mathsf{Setup}(1^\lambda, 1^D, |C|)$ procedure that selects public parameters $\mathsf{pp} := (1^D, G, p, g, \mathsf{Hash}, \delta, \gamma = \lceil |C|^{2/3} \cdot D^{1/3} \rceil)$ for all sub-protocols HRead, HCompress, HGate. [11]

- Sample a DDH instance $(G, p, g) \leftarrow \mathsf{Gen}(1^\lambda)$, with prime order p.
- Set $W = \lceil (|C|/D)^{1/3} \rceil$, and sample $b_i \leftarrow [p]$ for $i \in [W]$, which define a global hash function $\mathsf{Hash} : \mathbb{Z}^W \to \mathbb{Z}^{\lceil \log p \rceil}$ as specified in Equation 5.1.
- Set $\delta < 1/(p(\lambda) \cdot (3(D+1) + |C|))$ s.t. the probability over all invocations of the sub-protocols is $< 1/p(\lambda)$ for any desired polynomial $p(\lambda)$.
- Additionally, pre-compute values $\{g^{2^i}\}_{i \in [\lceil \log p \rceil]}$ for achieving low-depth exponentiations (by the garbler).

We next describe how to sample gk, and the $\mathsf{Enc}(\mathsf{gk}, \mathbf{x})$ procedure that produces encodings $[\![\mathbf{x}]\!]$ and public data pd.

- First choose leveled secrets $\mathbf{s}^{\mathsf{id}} \in [p]^2$ as gk for our encodings:

$$\mathbf{s}^{(0,\mathsf{hash})}, \mathbf{s}^{(0,\mathsf{OutWire})}, \mathbf{s}^{(D+1,\mathsf{InWire})} \leftarrow [p]^2,$$
$$\mathbf{s}^{(d,\mathsf{InWire})}, \mathbf{s}^{(d,\mathsf{OutWire})}, \mathbf{s}^{(d,\mathsf{hash})} \leftarrow [p]^2 \text{ for } d = 1, \ldots, D. \tag{5.3}$$

[11] In our garbling scheme definition, the Setup algorithm doesn't take $|C|$ or D as input. This syntactical mismatch can be solved by delaying settings of Hash, δ, γ which depends on $|C|$, D to GarbC, when the circuit is known.

- Next sets up public data for the sub-protocols:

$$\begin{aligned}
\mathsf{HW.pd}^{(d)} &\leftarrow \mathsf{Hash.Setup}(1^\lambda, \mathbf{s}^{\mathsf{id}}, \mathbf{s}^{\mathsf{id}'}),\ \mathsf{HR.pd}^{(d)} \leftarrow \mathsf{Hash.Setup}(1^\lambda, \mathbf{s}^{\mathsf{id}'}, \mathbf{s}^{id''}) \\
&\quad \text{where } \mathsf{id} = (d, \mathsf{OutWire}), \mathsf{id}' = (d, \mathsf{hash}), \mathsf{id}'' = (d+1, \mathsf{InWire}) \text{ for } d = 0, \ldots, D \\
\mathsf{GE.pd}^{(d)} &\leftarrow \mathsf{GE.Setup}(1^\lambda, \mathbf{s}^{\mathsf{id}}, \mathbf{s}^{\mathsf{id}'}) \\
&\quad \text{where } \mathsf{id} = (d, \mathsf{InWire}), \mathsf{id}' = (d, \mathsf{OutWire}) \text{ for } d = 1, \ldots, D \\
\mathsf{pd} &:= (\{\mathsf{HW.pd}^{(d)}\}, \{\mathsf{HR.pd}^{(d)}\}, \{\mathsf{GE.pd}^{(d)}\}).
\end{aligned} \tag{5.4}$$

- Finally use $\mathbf{s}^{\mathsf{id}=(0,\mathsf{OutWire})}$ from the above to compute

$$\begin{aligned}
\langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \rangle_G \leftarrow [p]^2, \quad \langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \rangle_E \leftarrow \langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \rangle_G + \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \text{ over } \mathbb{Z}_p, \\
[\![\mathbf{x}[i]]\!]_G := (\langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \rangle_G, \mathsf{id}), \quad [\![\mathbf{x}[i]]\!]_E := (\langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \rangle_E, \mathsf{id}, \mathbf{x}[i]).
\end{aligned} \tag{5.5}$$

Garbler Depth Complexity. As discussed in Section 4.1, the garbling depth roughly equals the sum of the depth of HRead, HCompress, HGate. We review the depth of each of the sub-protocols.

HRead**:** The garbler runs HRead_G, which consists of a call to $\mathsf{Extract}_G$.

$\mathsf{Extract}_G$ consists mainly of a summation over $\lceil \log p \rceil = O(\lambda)$ integers, each of $O(\lambda)$ bits, an exponentiation in G, a DDLog, and another summation over $O(\lambda)$ integers of $O(\lambda)$ bits. Assuming the group G is a subgroup of $\mathbb{Z}^*_{p'}$, then exponentiation can be computed with depth $O(\log \lambda)$ with pre-computated values $\{g^{2^i}\}_{i \in \lceil \log p \rceil}$. All of the other steps also have depth $O(\log \lambda)$ hence Extract has depth $O(\log \lambda)$.

We conclude the depth of HRead_G is $O(\log \lambda)$.

HCompress**:** The garbler runs $\mathsf{HCompress}_G$, which mainly consists of a call to Compress.

$\mathsf{Compress}_G$ consists of a SecSwitch and two summations of $O(\lambda)$ integers of $O(\lambda)$ bits. SecSwitch further consists of two exponentiations in G, one multiplication in G, and a DDLog. Therefore, similarly to $\mathsf{Extract}_G$, $\mathsf{Compress}_G$ has depth $O(\log \lambda)$.

We conclude the depth of HCompress is also $O(\log \lambda)$.

HGate**:** The garbler runs EvalKey, which mainly consists of a call to SecSwitch and MultKey. The construction of MultKey from [ILL25a] consists of constant number of exponentiations and multiplications in G, similarly to our SecSwitch construction, and hence has depth $O(\log \lambda)$.

Succinctness and Decomposability. The garbling material consists of the public data pd described in Equation 5.4, and a series of re-key materials $\{\mathsf{rk}^{(d)}\}_{d \in [D]}$. We briefly note the following, where we let $W = (|C|/D)^{1/3}$ denote the compression factor of Hash.

- The re-key material of each depth $\mathsf{rk}^{(d)}$ is of size $\lceil |C_d|/W \rceil \cdot \mathsf{poly}(\lambda)$, where $|C_d|$ is the number of depth-d wires. Overall, we have $|\{\mathsf{rk}^{(d)}\}_{d \in [D]}| \le (|C|/W + D) \cdot \mathsf{poly}(\lambda)$.
- The public data for sub-protocols $\mathsf{HR.pd}^{(d)}$, $\mathsf{HW.pd}^{(d)}$, $\mathsf{GE.pd}^{(d)}$ all have size $< W^2 \cdot \mathsf{poly}(\lambda)$. Overall the public data have total size $\le W^2 \cdot D \cdot \mathsf{poly}(\lambda)$.

Therefore, with our setting $W = (|C|/D)^{1/3}$ the overall garbling material have size $< (D + |C|^{2/3} \cdot D^{1/3}) \cdot \mathsf{poly}(\lambda)$.

The input encodings described in Equation 5.5 are clearly decomposable, and have size $< \ell \cdot \mathsf{poly}(\lambda)$. The decryption key dk, as specified in Definition 3.1, consists of a single integer of $O(\lambda) < \mathsf{poly}(\lambda)$ bits.

Remark 5.27. We note that in this work we choose to associate the encodings of wire values at each depth d a different secret. This allows us to base security on the standard DDH assumption, but also requires us to setup different instances, with fresh public data, of hashing Hash and gate evaluation GE schemes for each depth. Alternatively,

assuming suitable circular variants of DDH, we can let encodings of all wires values share a common secret, and setup only a single instance of SS, Hash, GE each. In this circular variant, the size of re-key materials remains bounded by $\leq (|C|/W + D) \cdot \mathsf{poly}(\lambda)$, while the size of public data is now bounded by $W^2 \cdot \mathsf{poly}(\lambda)$. Choosing $W = |C|^{1/3}$, the overall garbling material have size $\leq (D + |C|^{2/3}) \cdot \mathsf{poly}(\lambda)$. Similar analysis of garbling sizes also holds for our group instantiation of standard (private) garbling in Section 6.1.

Amortizing Large Public Data. We note that according to the above analysis, an alternative setting of the compression factor $W = |C|$ results in re-key material sizes $|\{\mathsf{rk}^{(d)}\}| \leq D \cdot \mathsf{poly}(\lambda)$, and pubilc data sizes $< |C|^2 \cdot D \cdot \mathsf{poly}(\lambda)$. While such a setting makes little sense in a single instance of garbling, where the size of garbling material is dominated by the size of public data, it can be useful in a scenario where the same garbler is to garble multiple circuits of the same depth. In such a scenario, the public data can be re-used across all circuits garbled by the same garbler, so its cost in garbling size is amortized. The amortized size of garbling material becomes $D \cdot \mathsf{poly}(\lambda)$.

In more detail, we rename the publie data to be evk, viewing it as an evaluation key for the designated garbler. We modify the garbling scheme syntax to have an explicit KeyGen algorithm for producing evk as follows.

- $\mathsf{Setup}(1^\lambda, 1^d) \to \mathsf{pp}$: Unchanged from Definition 3.1.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{evk}, \mathsf{msk})$: On input public parameters, the KeyGen algorithm outputs a pair of valuation key evk and master secret key msk.
- $\mathsf{GarbC}(\mathsf{msk}, C) \to (\hat{C}, \mathsf{gk})$: On input a master secret key and a circuit C (of depth $\leq d$), the GarbC algorithm outputs garbled circuit $\hat{C}$ and garbling key gk.
- $\mathsf{GarbInp}(\mathsf{gk}, x) \to \hat{x}$: Unchanged from Definition 3.1.
- $\mathsf{Eval}(\mathsf{evk}, C, \hat{C}, \hat{x}) \to \{0,1\}$: On input an evaluation key evk, circuit C, garbled circuit $\hat{C}$ and garbled input $\hat{x}$, the Eval algorithm computes the evaluation of the garbling circuit.

For correctness, we require for all polynomials $m(\cdot), p(\cdot), d(\cdot)$, there exists negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all circuits $\{C_i : \{0,1\}^{\ell_i} \to \{0,1\}\}_{i\in[m]}$ of size $|C_i| \leq p(\lambda)$ and depth $\leq d(\lambda)$, all inputs $\{x_i \in \{0,1\}^{\ell_i}\}_{i\in[m]}$,

$$\Pr\left[\begin{array}{c|c} \mathsf{Eval}(\mathsf{evk}, C_i, \hat{C}_i, \hat{x}_i) = C_i(x_i) & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d) \\ \forall i \in [m] & (\mathsf{evk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ & (\hat{C}_i, \mathsf{gk}_i) \leftarrow \mathsf{GarbC}(\mathsf{msk}, C_i) \\ & \hat{x}_i \leftarrow \mathsf{GarbInp}(\mathsf{gk}_i, x_i) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

For authenticity, we modify the security game $\mathsf{ExptAuth}_{\mathcal{A}}(\lambda)$ from Definition 3.1 as follows:

1. The adversary specifies a depth bound 1^d, an arbitrary number of circuits $\{C_i\}_{i\in[m]}$ (of depth $\leq d$), and corresponding inputs $\{x_i^*\}$. The challenger computes the following:
   (a) The challenger samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, and a pair of evaluation keys and master secret keys $(\mathsf{evk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$.
   (b) The challenger produces garbled circuits $(\hat{C}_i, \mathsf{gk}_i) \leftarrow \mathsf{GarbC}(\mathsf{msk}, C_i)$, garbled inputs $\hat{x^*}_i \leftarrow \mathsf{GarbInp}(\mathsf{gk}_i, x_i^*)$, and accompanying authentication keys auk_i^* for $i \in [m]$.
2. The challenger returns $(\mathsf{pp}, \mathsf{evk}, \{\hat{C}_i, \hat{x^*}_i = (L_{x^*,i}, \mathsf{dk}_i^*)\})$.
3. The adversary submits a label L_i^* and an index $i \in [m]$. The output of the experiment is 1 iff $1 - C_i(x_i^*) = \mathsf{DecodeLabel}(\mathsf{dk}_i^*, L_i^*)$ and $\mathsf{Authenticate}(\mathsf{auk}_i^*, L_i^*) = 1$.

Security. We sketch *selective* One-Wayness (Definition 3.2, with privacy error $\delta(\lambda) = 1/\mathsf{poly}(\lambda)$) for the garbling scheme via a series of hybrid experiments, where Hyb_0 is the experiment $\mathsf{ExptOW}_{\mathcal{A}}(\lambda)$, and Hyb_3 is an experiment where the adversary's advantage clearly negligible.

- Hyb_0: We briefly recall the experiment $\mathsf{ExptOW}_{\mathcal{A}}(\lambda)$:
  - The challenger returns public parameters pp according to Setup, and receives a circuit C and an input $\mathbf{x}$ from the adversary.
  - The challenger returns $\widehat{C}$, which consists of the public data pd as described in Equation 5.4, consisting of $\{\mathsf{HW.pd}^{(d)}\}$, $\{\mathsf{HR.pd}^{(d)}\}$, $\{\mathsf{GE.pd}^{(d)}\}$, and the rekey materials $\{\mathsf{rk}^{(d)}\}$.
  - The challenger also returns $\hat{\mathbf{x}}$, which consists of the encodings $[\![\mathbf{x}]\!]_E$ as described in Equation 5.5 and a decoding key dk that equals the last component of garbler's shares of the output wires.
- Hyb_1: This hybrid derive the rekey materials $\{\mathsf{rk}^{(d)}\}$ and the decoding key dk differently:
  - Using the input encodings $[\![\mathbf{x}]\!]_E$, derive evaluator's shares on all intermediate wires and hashes, and set $\{\mathsf{rk}^{(d)}\}$ as the difference between evaluator's shares of hashes instead of the garbler's.
  - Derive the decoding key dk as the last component of evaluator's shares of the output wires, subtracting $\mathbf{s}^{D+1,\mathsf{InWire}}[1] \cdot C(\mathbf{x})$.

  By the correctness of the garbling scheme, this hybrid is equivalent to the previous, except with $\delta(\lambda) = 1/\mathsf{poly}(\lambda)$ failure probability.
  $$|\Pr[\mathsf{Hyb}_{1,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}_{0,\mathcal{A}}(\lambda) = 1]| \leq \delta(\lambda).$$
- Hyb_2: This hybrid computes the input encodings as random elements in $\mathbb{Z}_q^2$, instead of following Equation 5.5. This hybrid is equivalent to the previous, by the randomness of $\langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \rangle_G$. (See Equation 5.5.)
  $$\Pr[\mathsf{Hyb}_{2,\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}_{1,\mathcal{A}}(\lambda) = 1].$$
  We note that the secret $\mathbf{s}^{(0,\mathsf{OutWire})}$ is only used for generatring public data $\mathsf{HW.pd}^{(0)}$, and not anywhere else in the experiment.
- Hyb_3: This hybrid samples the public data $\{\mathsf{HW.pd}^{(d)}\}$, $\{\mathsf{HR.pd}^{(d)}\}$, $\{\mathsf{GE.pd}^{(d)}\}$, as random group elements in G, instead of as a chain of ciphertexts encrypting the next secret s' under some secret s. By Lemma 5.21 and 5.26, both of which assumes the DDH assumption in prime-order groups, we have
  $$|\Pr[\mathsf{Hyb}_{3,\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}_{2,\mathcal{A}}(\lambda) = 1]| < \mathsf{negl}(\lambda).$$
  We note that the first component $\mathbf{s}^{(D+1,\mathsf{InWire})}[0]$ of the secret for the output label is no longer used anywhere in the experiment. So the probability of the adversary guessing this random secret is negligible, i.e. $\Pr[\mathsf{Hyb}_{3,\mathcal{A}}(\lambda) = 1] < \mathsf{negl}(\lambda)$.

By a standard hybrid argument, we have
$$\Pr[\mathsf{ExptOW}_{\mathcal{A}}(\lambda) = 1] < \Pr[\mathsf{Hyb}_{3,\mathcal{A}}(\lambda) = 1] + \mathsf{negl}(\lambda) + \delta(\lambda) = \mathsf{negl}(\lambda) + \delta(\lambda).$$

Removing the $1/\mathsf{poly}(\lambda)$ Correctness and Privacy Error. We note two ways to remove the correctness error, which also causes the privacy error, from our group instantiation. First, the simpler solution is to use Pailler instead of prime-order groups, which has an associated DDLog algorithm with negligible error. The Paillier instantiation relies on the corresponding Paillier-based aHMAC evaluation algorithms from [ILL25a] for HGate. And the constructions for HCompress, HRead sub-protocols are analogous to those in prime-order groups. Security of the obtained garbling scheme relies on the DDH assumption in Paillier groups (Definition 5.9).

Second, we can adapt the solutions from [ILL25b, BGI17] for prime-order groups to achieve negligible correctness error at the cost of heavier computation. In more detail, we rely on the fact that both parties running the DDLog

algorithm can locally detect whether a failure has occurred (with false positives). Therefore, the garbler can abort and restart as soon as a failure during DDLog is detected. When setting this failure probability to be $1/\lambda$, the garbler is expected to restart $O(\lambda)$ times. Note that all other steps besides DDLog has negligible failure probability, the final garbled circuit produced by the garbler achieves negligible correctness error. Security still relies on the standard DDH assumption.

We note a subtlety in privacy: the restarting pattern by the garbler depends on the (leveled) secrets $\mathbf{s}^{\mathsf{id}}$ in our constructions, and hence leaks some information about them. Following the solutions from [ILL25b, BGI17], we compensate by deriving $\mathbf{s}$ from linearly combining $O(\lambda \cdot \log p)$ secret bits, and ensuring that at most λ bits are affected by the leakage. We refer to [ILL25b, BGI17] for details of dealing with such leakages.

5.3 Instantiation Under Lattices

Similarly to the previous section, our lattice instantiation uses the aHMAC evaluation algorithms EvalKey, EvalTag based on RLWE from [ILL25c] to implement the HGate protocol, and our main focus is to design compatible protocols HRead and HCompress.

In more detail, in our lattice instantiation, the encoding of a wire value $[\![x]\!]$ for $x \in \{0,1\}$ contains a pair of additive shares $\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle$ over $\mathcal{R}_p$, where $\mathcal{R} = \mathbb{Z}[X]/(X^n+1)$ is a polynomial ring, p is a modulus, and $\mathbf{s}^{\mathsf{id}} \in \mathcal{R}^2$ are two secret elements with an associated id. Again, the value x is revealed to the evaluator.

$$[\![x]\!] = ([\![x]\!]_G, [\![x]\!]_E), \text{ where } [\![x]\!]_G = (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_G, \mathsf{id}), \text{ and } [\![x]\!]_E = (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_E, \mathsf{id}, x).$$

The $\mathsf{id} = (d, \mathsf{type})$ is defined in the same way as the previous section.

Under this encoding, the Rekey steps are as follows, assuming access to public parameters pp that specify the ring $\mathcal{R}$ and modulus p.

- $\mathsf{RekeySamp}(\mathsf{id})$ samples random share $\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_G \leftarrow \mathcal{R}_p^2$, and sets $[\![x']\!]_G := (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_G, \mathsf{id})$.
- $\mathsf{RekeyGen}([\![x]\!]_G, [\![x']\!]_G)$ outputs the difference $\mathsf{rk} \leftarrow \langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_G - \langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_G$.
- $\mathsf{RekeyEval}([\![x]\!]_E, \mathsf{rk})$ computes $\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_E \leftarrow \langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle_E + \mathsf{rk}$, and sets $[\![x']\!]_E := (\langle \mathbf{s}^{\mathsf{id}} \cdot x \rangle'_E, \mathsf{id}, x)$.

Instantiation of HRead and HCompress. The sub-protocols HRead, HCompress in our lattice instantiation are the same as those in the previous section (Figure 1, and 2), except they use the local algorithms $\mathsf{HRead}_G, \mathsf{HRead}_E$, $\mathsf{HCompress}_G, \mathsf{HCompress}_E$ from Construction 5.33 We present these new local algorithms below.

Similarly to the previous section, we first present the SecSwitch algorithm (Construction 5.28) as a building block, and next our core construction of a pair of fixed-factor hashing algorithms (Construction 5.30). Then, we describe wrapper algorithms $\mathsf{HRead}_G, \mathsf{HRead}_E, \mathsf{HCompress}_G, \mathsf{HCompress}_E$ that call the fixed-factor hashing in Construction 5.33.

We note a technical difference in constructing the wrapper algorithms. While in the group instantiation (Construction 5.18), the wrapper algorithms make parallel calls to fixed-factor hashing, in the lattice instantiation the wrapper algorithms instead make recursive calls. At a high-level, the difference stems from the fact that the hash function in the group instantiation, defined by W coefficients (Equation 5.1), is private to the evaluator, while the hash function in the lattice instantiation (Equation 5.6) is public. The benefit of using recursive, instead of parallel, calls to fixed factor hashing is that we can now achieve a much larger compression factor, and hence much smaller hashes. Consequently, our garbling scheme has much smaller communication cost from rekeying the hashes. (See Theorem 5.2.)

Construction 5.28 (Secret Switching for aHMAC Encodings)**.** The construction is with respect to the following public parameters pp:

- A bound $B < 2^{\mathsf{poly}(\lambda)}$ on encoded values;
- A polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n+1)$ where $n < \mathsf{poly}(\lambda)$ is a power-of-two;
- Two moduli $p > B \cdot \lambda^{\omega(1)}$, and $q = p \cdot \Delta$, where $\Delta > B \cdot p \cdot \lambda^{\omega(1)}$.

- Two distributions $\mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}}$ over $\mathcal{R}$ with coefficients bounded by $\mathsf{poly}(\lambda)$.

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s}, \mathbf{s}' \in \mathcal{D}_{\mathsf{sk}}^2)$**:** Compute the secret-switching public data as

$$a \leftarrow \mathcal{R}_q,\ \mathbf{e} \leftarrow \mathcal{D}_{\mathsf{err}}^2, b \leftarrow \mathbf{s} \cdot a + \mathbf{e} + \mathbf{s}' \cdot \Delta \text{ over } \mathcal{R}_q$$
$$\mathsf{pd} := (a, b, \alpha, \beta), \text{ where } \alpha \leftarrow \mathcal{R}_q^2,\ \beta \leftarrow \mathcal{R}_p^2.$$

$\langle \mathbf{s}' \cdot x \rangle_{G/E} \leftarrow \mathsf{SecSwitch}^{\mathsf{pd}}(\langle \mathbf{s} \cdot x \rangle_{G/E}, \langle x \rangle_{G/E})$**:** Read (a, b) from pd and compute the following

$$\begin{aligned}
\langle \mathbf{s} \cdot x \rangle'_{G/E} &\leftarrow \langle \mathbf{s} \cdot x \rangle_{G/E} + \beta,\ \langle x \rangle'_{G/E} \leftarrow \langle x \rangle_{G/E} + \beta, && \text{over } \mathcal{R}_p, \\
\langle \mathbf{s}' \cdot x \rangle'_{G/E} &\leftarrow a \cdot \langle \mathbf{s} \cdot x \rangle'_{G/E} - b \cdot \langle x \rangle'_{G/E} + \alpha && \text{over } \mathcal{R}_q, \\
\langle \mathbf{s}' \cdot x \rangle_{G/E} &\leftarrow \lfloor \langle \mathbf{s}' \cdot x \rangle'_{G/E} / \Delta \rfloor && \text{over } \mathcal{R}_p.
\end{aligned}$$

We show the following correctness lemma.

Lemma 5.29 (Correctness of Secret Switching)**.** *There exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$, public parameters* pp *as specified, encoded value $x \in \mathcal{R}_B$, secrets $\mathbf{s}, \mathbf{s}' \in \mathcal{D}_{\mathsf{sk}}$, additve shares $\langle \mathbf{s} \cdot x \rangle$, the following holds:*

$$\Pr\left[\langle \mathbf{s}' \cdot x \rangle_E = \langle \mathbf{s}' \cdot x \rangle_G + \mathbf{s}' \cdot x \,\middle|\, \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \cdot x \rangle_G \leftarrow \mathsf{SecSwitch}(\langle \mathbf{s} \cdot x \rangle_G), \\ \langle \mathbf{s}' \cdot x \rangle_E \leftarrow \mathsf{SecSwitch}(\langle \mathbf{s} \cdot x \rangle_E) \end{array}\right] > 1 - \mathsf{negl}(\lambda)$$

Proof. We first claim that the intermediate values $\langle \mathbf{s} \cdot x \rangle'_{G/E}, \langle x \rangle'_{G/E}$ during SecSwitch form additive shares of $\mathbf{s} \cdot x$ and x *over* $\mathcal{R}$. This holds with except probability $O(n \cdot \|\mathbf{s} \cdot x\|_\infty / p)$ (over the randomness of β) by the lifting lemma (Lemma 2) of [BKS19]. By our parameter setting, we have

$$n \cdot \|\mathbf{s} \cdot x\|_\infty \le \mathsf{poly}(\lambda) \cdot B \ll p.$$

So the failure probability is negligible. We next check that the $\langle \mathbf{s}' \cdot x \rangle'_{G/E}$ values form additve shares of $\mathbf{s}' \cdot x \cdot \Delta + \mathsf{err}$ over $\mathcal{R}_q$.

$$\begin{aligned}
\langle \mathbf{s}' \cdot x \rangle'_E &= a \cdot \langle \mathbf{s} \cdot x \rangle'_E - b \cdot \langle x \rangle'_E + \alpha \\
\text{(By Construction and Above)} &= \langle \mathbf{s}' \cdot x \rangle'_G + a \cdot \mathbf{s} \cdot x - b \cdot x \\
\text{(By Setup)} &= \langle \mathbf{s}' \cdot x \rangle'_G + a \cdot \mathbf{s} \cdot x - (\mathbf{s} \cdot a + \mathbf{e} + \mathbf{s}' \cdot \Delta) \cdot x \\
&= \langle \mathbf{s}' \cdot x \rangle'_G + \mathbf{s}' \cdot x \cdot \Delta + \mathsf{err},
\end{aligned}$$

where $\|\mathsf{err}\|_\infty < \mathsf{poly}(\lambda) \cdot B$. Finally, by the rounding lemma (Lemma 1) of [BKS19], the derived values $\langle \mathbf{s}' \cdot x \rangle_{G/E}$ form additive shares of $\mathbf{s}' \cdot x$ as desired, except with probability $O(n \cdot \|\mathsf{err}\| / \Delta)$ (over the randomness of α). By our parameter setting, we have

$$n \cdot \|\mathsf{err}\| \le \mathsf{poly}(\lambda) \cdot B \ll \Delta.$$

So correctness of SecSwitch holds except with negligible probability. □

Construction 5.30 (Fixed Factor Hashing for aHMAC Encodings)**.** The construction is with respect to the same public parameters $(B, \mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ as specified in Construction 5.28, and additionally:

- A distribution $\mathcal{D}_{\mathsf{pub}}$ (over $\mathcal{R}$) of elements with uniform binary coefficients.
- A hash function $\mathsf{Hash} : \mathcal{R}^{W \cdot \lceil \log q \rceil} \to \mathcal{R}^{\lceil \log q \rceil}$ with compression factor $W < \mathsf{poly}(\lambda)$, defined by $W' = W \cdot \lceil \log q \rceil$ ring elements $\mathbf{b}_i \in \mathcal{D}_{\mathsf{pub}}^{\lceil \log q \rceil}$ for $i \in [W']$;

$$\mathsf{Hash}(\mathbf{x} \in \mathcal{R}_B^{W'}) := \sum_i \mathbf{b}_i \cdot \mathbf{x}[i] \text{ over } \mathcal{R}. \tag{5.6}$$

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s}, \mathbf{s}' \in \mathcal{D}_{\mathsf{sk}}^2)$**:** Read $\{\mathbf{b}_i\}_{i\in[W']}$ from pp, and generate ciphertexts $\{\mathsf{ct}_{i,j,b}\}_{i,j\in[W'],b\in\{0,1\}}$ used for extracting hashed values:

$$\begin{aligned}\mathbf{a}_j &\leftarrow \mathcal{R}_q^{\lceil \log q\rceil},\ \mathbf{e}_j \leftarrow \mathcal{D}_{\mathsf{err}}^{\lceil \log q\rceil},\ e_{i,j} \leftarrow \mathcal{D}_{\mathsf{err}}\\ \mathsf{ct}_{i,j,b} &\leftarrow (\mathbf{s}[b]\cdot\mathbf{a}_j + \mathbf{e}_j)^\intercal \cdot \mathbf{b}_i + e_{i,j} - \mathbf{s}'[b]\cdot\mathbf{u}_j[i]\cdot\Delta \text{ over } \mathcal{R}_q,\end{aligned}$$

where $\mathbf{u}_j \in \{0,1\}^{W'}$ denote the j-th unit vector. Finally, sample random offsets α, β, and output

$$\mathsf{pd} := (\mathsf{pp}, \{\mathsf{ct}_{i,j,b}\}, \{\mathbf{a}_j\}, \alpha, \beta),\ \text{where } \alpha \leftarrow \mathcal{R}_q^2,\ \beta \leftarrow \mathcal{R}_p^{2\cdot\lceil\log q\rceil}.$$

$\langle \mathbf{s}\otimes\mathbf{h}\rangle_{G/E} \leftarrow \mathsf{Compress}^{\mathsf{pd}}(\langle \mathbf{s}\otimes\mathbf{x}\rangle_{G/E})$**:** Read $\{\mathbf{b}_i\}_{i\in[W']}$ from pd, and compute

$$\langle \mathbf{s}\otimes\mathbf{h}\rangle_{G/E} \leftarrow \sum_i \langle \mathbf{s}\cdot\mathbf{x}[i]\rangle_{G/E}\otimes\mathbf{b}_i \text{ over } \mathcal{R}_p.$$

$\langle \mathbf{s}'\cdot x_j\rangle_G \leftarrow \mathsf{Extract}_G^{\mathsf{pd}}(\langle \mathbf{s}\otimes\mathbf{h}\rangle_G, j\in[W])$**:** Read $\mathbf{a}_j \in \mathcal{R}_q^{\lceil\log q\rceil}$ and $\alpha\in\mathcal{R}_q^2, \beta\in\mathcal{R}_p^{2\cdot\lceil\log q\rceil}$ from pd, and compute

$$\begin{aligned}\langle \mathbf{s}\otimes\mathbf{h}\rangle'_G &\leftarrow \langle \mathbf{s}\otimes\mathbf{h}\rangle_G + \beta && \text{over } \mathcal{R}_p,\\ \langle \mathbf{s}'\cdot x_j\rangle'_G &\leftarrow \left(\sum_{k\in[\lceil\log q\rceil]} \mathbf{a}_j[k]\cdot\langle \mathbf{s}\otimes\mathbf{h}[k]\rangle'_G\right) + \alpha && \text{over } \mathcal{R}_q,\\ \langle \mathbf{s}'\cdot x_j\rangle_G &\leftarrow \lfloor \langle \mathbf{s}\cdot x_j\rangle'_G/\Delta\rfloor && \text{over } \mathcal{R}_p.\end{aligned}$$

$\langle \mathbf{s}'\cdot x_j\rangle_E \leftarrow \mathsf{Extract}_E^{\mathsf{pd}}(\langle \mathbf{s}\otimes\mathbf{h}\rangle_E, j, \mathbf{x}\in\mathcal{R}_B^{W'})$**:** First read $\{\mathsf{ct}_{i,j,b}\}_{i,j\in[W'],b\in\{0,1\}}$ from pd, and compute

$$\mathsf{diff}_j[b] \leftarrow \sum_i \mathsf{ct}_{i,j,b}\cdot\mathbf{x}[i] \text{ over } \mathcal{R}_q.$$

Next read $\alpha\in\mathcal{R}_q^2, \beta\in\mathcal{R}_p^{2\cdot\lceil\log q\rceil}$ from pd, and compute

$$\begin{aligned}\langle \mathbf{s}\otimes\mathbf{h}\rangle'_E &\leftarrow \langle \mathbf{s}\otimes\mathbf{h}\rangle'_E + \beta && \text{over } \mathcal{R}_p,\\ \langle \mathbf{s}'\cdot x_j\rangle'_E &\leftarrow \left(\sum_{k\in[\lceil\log q\rceil]} \mathbf{a}_j[k]\cdot\langle \mathbf{s}\otimes\mathbf{h}[k]\rangle'_E\right) - \mathsf{diif}_j + \alpha && \text{over } \mathcal{R}_q,\\ \langle \mathbf{s}'\cdot x_j\rangle_E &\leftarrow \lfloor \langle \mathbf{s}\cdot x_j\rangle'_E/\Delta\rfloor && \text{over } \mathcal{R}_p.\end{aligned}$$

We show the following correctness lemma.

Lemma 5.31 (Correctness of Fixed Factor Hashing)**.** *There exists a negligible function* negl *such that for every $\lambda\in\mathbb{N}$, public parameters* pp *as specified, encoded vectors $\mathbf{x}\in\mathcal{R}_B^{W'}$, secrets $\mathbf{s},\mathbf{s}'\in\mathcal{D}_{\mathsf{sk}}^2$, additve shares $\langle\mathbf{s}\otimes\mathbf{x}\rangle$, $\langle\mathbf{s}\cdot\mathsf{Hash}(\mathbf{x})\rangle$, and index $j\in[W']$, the following holds:*

$$\Pr\left[\langle\mathbf{s}\otimes\mathbf{h}\rangle_E = \langle\mathbf{s}\otimes\mathbf{h}\rangle_G + \mathbf{s}\otimes\mathsf{Hash}(\mathbf{x}) \;\middle|\; \begin{array}{c}\mathsf{pd}\leftarrow\mathsf{Setup}(1^\lambda,\mathbf{s},\mathbf{s}'),\\ \langle\mathbf{s}\otimes\mathbf{h}\rangle_G\leftarrow\mathsf{Compress}(\langle\mathbf{s}\otimes\mathbf{x}\rangle_G),\\ \langle\mathbf{s}\otimes\mathbf{h}\rangle_E\leftarrow\mathsf{Compress}(\langle\mathbf{s}\otimes\mathbf{x}\rangle_E)\end{array}\right] > 1-\mathsf{negl}(\lambda)$$

$$\Pr\left[\langle\mathbf{s}'\cdot x_j\rangle_E = \langle\mathbf{s}'\cdot x_j\rangle_G + \mathbf{s}'\cdot\mathbf{x}[j] \;\middle|\; \begin{array}{c}\mathsf{pd}\leftarrow\mathsf{Setup}(1^\lambda,\mathbf{s},\mathbf{s}'),\\ \langle\mathbf{s}'\cdot x_j\rangle_G\leftarrow\mathsf{Extract}_G(\langle\mathbf{s}\otimes\mathsf{Hash}(\mathbf{x})\rangle_G, j),\\ \langle\mathbf{s}'\cdot x_j\rangle_E\leftarrow\mathsf{Extract}_E(\langle\mathbf{s}\otimes\mathsf{Hash}(\mathbf{x})\rangle_E, j, \mathbf{x}),\end{array}\right] > 1-\mathsf{negl}(\lambda)$$

Proof. The correctness of Compress follows directly from the definition of Hash. We verify correctness of Extract. First, by the lifting lemma from [BKS19] and analogous to the proof of Lemma 5.29, the intermediate values $\langle \mathbf{s} \otimes \mathbf{h} \rangle'_{G/E}$ form additive shares of $\mathbf{s} \otimes \mathbf{h}$ over $\mathcal{R}$ (and hence over $\mathcal{R}_q$), except with negligible probability. We show that the values $\langle \mathbf{s}' \cdot x_j \rangle'_{G/E}$ form additive shares of $\mathbf{s}' \cdot x_j \cdot \Delta + \text{err}$ over $\mathcal{R}_q$.

$$
\begin{aligned}
\langle \mathbf{s}' \cdot x_j \rangle'_E &:= \sum_k \mathbf{a}_j[k] \cdot \langle \mathbf{s} \otimes \mathbf{h}[k] \rangle'_E - \text{diff}_j + \alpha \\
\text{(By above)} &= \sum_k \mathbf{a}_j[k] \cdot \left(\langle \mathbf{s} \otimes \mathbf{h}[k] \rangle'_G + \mathbf{s} \otimes \mathbf{h}[k] \right) - \text{diff}_j + \alpha \\
&= \langle \mathbf{s}' \cdot x_j \rangle'_G + \mathbf{s} \cdot \mathbf{a}_j^\intercal \cdot \mathbf{h} - \text{diff}_j \\
\text{(By Setup)} &= \langle \mathbf{s}' \cdot x_j \rangle'_G + \mathbf{s} \cdot \mathbf{a}_j^\intercal \cdot \mathbf{h} - \sum_i \mathbf{s} \cdot (\mathbf{a}_j^\intercal \cdot \mathbf{b}_i) \cdot \mathbf{x}[i] \\
&\qquad + \sum_i (\mathbf{e}_j^\intercal \cdot \mathbf{b}_i - e_{i,j}) \cdot \mathbf{x}[i] + \sum_i \mathbf{s}' \cdot \mathbf{u}_j[i] \cdot \Delta \cdot \mathbf{x}[i] \\
\text{(By Def. of Hash)} &= \langle \mathbf{s}' \cdot x_j \rangle'_G + \mathbf{s}' \cdot \mathbf{x}[j] \cdot \Delta + \text{err},
\end{aligned}
$$

where $\|\text{err}\|_\infty < \text{poly}(\lambda) \cdot B$. By the rounding lemma from [BKS19] and analogous to the proof of Lemma 5.29 we conclude that the values $\langle \mathbf{s}' \cdot x_j \rangle_{G/E}$ form additive share of $\mathbf{s}' \cdot x_j$ as desired, except with negligible probability. □

For our security proof later, and similarly to the group instantiation (Section 5.2) we show the following lemma.

Lemma 5.32 (Security of Fixed Factor Hashing)**.** *Under the RLWE assumption, every efficient adversary $\mathcal{A}$ has negligible advantage in distinguishing the following experiments $\text{Exp}^b_{\mathcal{A}}(1^\lambda)$, for $b \in \{0, 1\}$*

- *Sample public parameters* pp*, and sends them to $\mathcal{A}$, who returns exponents $\mathbf{s}' \in \mathcal{D}^2_{\text{sk}}$ and an arbitrary number of $\{\mathbf{s}^{(i)} \in \mathcal{D}^2_{\text{sk}}\}_i$.*
- *When $b = 0$, sample random exponents $\mathbf{s} \leftarrow \mathcal{D}^2_{\text{sk}}$ and send* pd, $\{\text{pd}^{(i)}\}$ *to $\mathcal{A}$, computed as follows*

$$\text{pd} \leftarrow \text{FH.Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'),\ \text{pd}^{(i)} \leftarrow \text{SS.Setup}(1^\lambda, \mathbf{s}, \mathbf{s}^{(i)}).$$

- *When $b = 1$, sample* pd, $\{\text{pd}^{(i)}\}$ *with random $\mathcal{R}_q$ elements and send them to $\mathcal{A}$.*

Proof. We sketch three hybrid experiments, with $\text{Hyb}_0 \equiv \text{Exp}^0_{\mathcal{A}}(1^\lambda)$, and $\text{Hyb}_2 \equiv \text{Exp}^1_{\mathcal{A}}(1^\lambda)$.

- Hyb_0: In the experiment $\text{Exp}^0_{\mathcal{A}}(1^\lambda)$, pd consists of an RLWE ciphertext of the form $a, b := \mathbf{s} \cdot a + \mathbf{e} + f(\mathbf{s}')$, for some function f, and $\text{pd}^{(i)}$ consists of RLWE ciphertexts of the form $a^{(i)}, \mathbf{c}^{(i)} := \mathbf{r}^{(i),\intercal} \cdot \mathbf{b}^{(i)} + \mathbf{e}^{(i)} + f^{(i)}(\mathbf{s}^{(i)})$ for some functions $f^{(i)}$, and where $\mathbf{r}^{(i)} := \mathbf{s} \otimes \mathbf{a}^{(i)} + \mathbf{E}^{(i)}$.
- Hyb_1: Sample the b element and $\mathbf{r}^{(i)}$ vectors from the above as random $\mathcal{R}_q$ elements instead. By RLWE, we have $\text{Hyb}_1 \approx_c \text{Hyb}_0$.
- Hyb_2: Sample the $\mathbf{c}^{(i)}$ vectors as random $\mathcal{R}_q$ elements instead. Note that the terms $\mathbf{r}^{(i),\intercal} \cdot \mathbf{b}^{(i)} + \mathbf{e}^{(i)}$, where $\mathbf{r}^{(i)} \leftarrow \mathcal{R}_q^{\lceil \log q \rceil}$ and $\mathbf{b}^{(i)} \leftarrow \mathcal{D}^{\lceil \log q \rceil}_{\text{pub}}$, form non-uniform (R)LWE samples as defined in [BLMR13]. Adopting the arguments there for LWE to the RLWE setting, we can conclude that such non-uniform RLWE samples are pseudorandom under the standard RLWE assumption. We have $\text{Hyb}_2 \approx_c \text{Hyb}_1$.

A standard hybrid argument then concludes the proof. □

Construction 5.33 (Hashing for aHMAC Encodings)**.** The construction is with respect to the following public parameters pp:

- A bound γ on the level of recursive compression.

- A polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where $n < \mathsf{poly}(\lambda)$ is a power-of-two;
- Two distributions $\mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}}$ over $\mathcal{R}$ with coefficients bounded by $\mathsf{poly}(\lambda)$.
- A hash function $\mathsf{Hash} : \mathcal{R}^{W'} \to \mathcal{R}^{\lceil \log q \rceil}$ defined as in Equation 5.6 for $W' = W\lceil \log q \rceil$, and $W = \lambda$.

It uses the secret switching algorithms (Construction 5.28, denoted with prefix SS) and the fixed factor hashing algorithms (Construction 5.30, denoted with prefix FH) as ingredients, with respect to the above and the following additional parameters.

- The $\mathcal{D}_{\mathsf{pub}}$ distribution as specified in Construction 5.30.
- A bound $B = (n \cdot W')^\gamma$ on encoded values.
- Two moduli $p = B \cdot \lambda^{\omega(1)}$, and $q = p \cdot \Delta$, where $\Delta = B \cdot p \cdot \lambda^{\omega(1)}$.

For a string $\mathbf{x} \in \{0,1\}^\ell$ of length $\ell < W^\gamma$, we define its hash $h_{\mathbf{x}}$ to be the (γ level) recursively hashed value of $\mathbf{x}$ by Hash (Equation 5.6), viewing each bit x_i as a ring element. For an index $i < W^\gamma$, we define $\mathsf{path}_i \in [W]^\gamma$ to be a sequence of indices corresponding to $\mathbf{x}[i]$ during recursive hashing.

$$\begin{aligned} \mathsf{path}_i[0] &= i \bmod W', \\ \mathsf{path}_i[t] &= \lfloor i/(\lceil \log p \rceil \cdot W^t) \rfloor \bmod W \text{ for } t \in [\gamma], t > 0 \end{aligned} \tag{5.7}$$

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s}, \mathbf{s}' \in \mathcal{D}_{\mathsf{sk}}^2)$**:** Sample secrets $\{\mathbf{s}^{(t)}\}_{t \in [\gamma+1]}$, prepare public data for γ instances of fixed factor hashing, and an for secret switching from $\mathbf{s}$ to $\mathbf{s}'$:

$$\begin{aligned} \mathbf{s}^{(0)}, \leftarrow \mathbf{s},\ \mathbf{s}^{(\gamma)} \leftarrow \mathbf{s}',\ \mathbf{s}^{(t)} \leftarrow \mathcal{D}_{\mathsf{sk}}^2 \text{ for } t = 1, \ldots, \gamma - 1 \\ \mathsf{FH.pd}^{(t)} \leftarrow \mathsf{FH.Setup}(1^\lambda, \mathbf{s}^{(t)}, \mathbf{s}^{(t+1)}) \text{ for } t \in [\gamma], \\ \mathsf{SS.pd} \leftarrow \mathsf{SS.Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'). \end{aligned}$$

Output $\mathsf{pd} := (\mathsf{SS.pd}, \{\mathsf{FH.pd}^{(t)}\}_{t \in [\gamma]})$.

$\langle \mathbf{s}' \cdot x_i \rangle_G \leftarrow \mathsf{HRead}_G^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{h} \rangle_G, i)$**:** Define path_i as in Equation 5.7, and assuming $i < W^\gamma$. Let $\langle \mathbf{s}^{(0)} \otimes \mathbf{h}^{(\gamma)} \rangle_G \leftarrow \langle \mathbf{s} \otimes \mathbf{h} \rangle_G$, and proceed as follows for $j \in [\gamma - 1]$:

- Read $\mathsf{FH.pd}^{(j)}$ from pd, and compute for $k \in [\lceil \log q \rceil]$

$$\langle \mathbf{s}^{(j+1)} \cdot \mathbf{h}^{(\gamma-j-1)}[k] \rangle_G \leftarrow \mathsf{FH.Extract}_G^{\mathsf{FH.pd}^{(j)}}(\langle \mathbf{s}^{(j)} \otimes \mathbf{h}^{(\gamma-j)} \rangle_G, \mathsf{path}_i[\gamma - j - 1] \cdot \lceil \log q \rceil + k). \tag{5.8}$$

Finally, compute $\langle \mathbf{s}' \cdot x_i \rangle_G := \langle \mathbf{s}^{(\gamma)} \cdot x_i \rangle_G \leftarrow \mathsf{FH.Extract}_G^{\mathsf{FH.pd}^{(\gamma-1)}}(\langle \mathbf{s}^{(\gamma-1)} \otimes \mathbf{h}^{(1)} \rangle_G, \mathsf{path}_i[0])$.

$\langle \mathbf{s}' \cdot x_i \rangle_E \leftarrow \mathsf{HRead}_E^{\mathsf{pd}}(\langle \mathbf{s} \cdot h \rangle_E, i, \mathbf{x})$**:** Analogous to HRead_G, except replacing Equation 5.8 with the following:

$$\begin{aligned} \langle \mathbf{s}^{(j+1)} \cdot \mathbf{h}^{(\gamma-j-1)}[k] \rangle_E &\leftarrow \mathsf{FH.Extract}_E^{\mathsf{FH.pd}^{(j)}}(\langle \mathbf{s}^{(j)} \otimes h^{(\gamma-j)} \rangle_E, \mathsf{path}_i[\gamma - j - 1] \cdot \lceil \log q \rceil + k, \boxed{\overline{\mathbf{h}}^{(\gamma-j)}}), \\ \langle \mathbf{s}^{(\gamma)} \cdot x_i \rangle_E &\leftarrow \mathsf{FH.Extract}_E^{\mathsf{FH.pd}^{(j)}}(\langle \mathbf{s}^{(j)} \cdot h^{(\gamma-j)} \rangle_E, \mathsf{path}_i[0], \boxed{\overline{\mathbf{h}}^{(1)}}), \end{aligned}$$

where $\overline{\mathbf{h}}^{(t-j)} \in \mathcal{R}^{W'}$ is the pre-image of the intermediate hash $h^{(t-j)}$, which can be derived from the input $\mathbf{x}$.

$\langle \mathbf{s}' \otimes \mathbf{h} \rangle_G \leftarrow \mathsf{HCompress}_G^{\mathsf{pd}}(\langle \mathbf{s} \otimes \mathbf{x} \rangle_G)$**:** Read SS.pd from pd. For every index $i \in |\mathbf{x}|$ run SecSwitch to switch the encoding secret from $\mathbf{s}$ to $\mathbf{s}'$:

$$\langle \mathbf{s}' \cdot \mathbf{x}[i] \rangle_G \leftarrow \mathsf{SS.SecSwitch}^{\mathsf{SS.pd}}(\langle \mathbf{s} \cdot \mathbf{x}[i] \rangle_G, \langle \mathbf{x}[i] \rangle_G), \text{ where } \langle \mathbf{x}[i] \rangle_G \leftarrow 0,$$

Next, recursively compress the $\langle \mathbf{s}' \otimes \mathsf{diff} \rangle_G$ by running FH.Compress. Proceed as follows for $j = [\gamma]$:

- Read $\mathsf{FH.pd}^{(j)}$ from pd, and compute the next level hash $\langle \mathbf{s}' \otimes \mathbf{h}^{(j+1)} \rangle_G$ as

$$\langle \mathbf{s}' \otimes \mathbf{h}^{(j+1)} \rangle_G[i] \leftarrow \mathsf{FH.Compress}_G^{\mathsf{FH.pd}^{(j)}}(\langle \mathbf{s}' \otimes \mathbf{h}_i^{(j)} \rangle_G),$$
$$\text{where } \mathbf{h}_{\mathrm{i}}^{(\mathrm{j})} := \mathrm{flatten}(\mathbf{h}^{(\mathrm{j})})[\mathrm{i} \cdot \mathrm{W}' : (\mathrm{i}+1) \cdot \mathrm{W}'].$$

  Finally, output the obatined $\langle \mathbf{s}' \otimes \mathbf{h}^{(\gamma)} \rangle_G$ as $\langle \mathbf{s}' \otimes \mathbf{h} \rangle_G$.

$\langle \mathbf{s}' \otimes \mathbf{h} \rangle_E \leftarrow \mathsf{HCompress}_E^{\mathsf{pd}}(\langle \mathbf{s} \cdot \mathbf{x} \rangle_E, \mathbf{x})$**:** Analogous to HCompress, except when running SS.SecSwitch, set $\langle \mathbf{x}[i] \rangle_E \leftarrow \mathbf{x}[i]$, and when running $\mathsf{FH.Compress}_E$, set $\langle \mathbf{h}_i^{(j)} \rangle_E \leftarrow \mathbf{h}_i^{(j)}$ where $\mathbf{h}_i^{(j)}$ is derived from the input $\mathbf{x}$ by recursive hashing.

We note the following correctness and security lemmas. The security lemma follows from applying Lemma 5.32 γ times.

Lemma 5.34 (Correctness of Hashing)**.** *There exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$, public parameters* pp *as specified, encoded vectors $\mathbf{x} \in \{0,1\}^{W^\gamma}$, index $j \in [W^\gamma]$, secrets $\mathbf{s}, \mathbf{s}' \in \mathcal{D}_{\mathsf{sk}}^2$, additve shares $\langle \mathbf{s} \cdot \mathbf{x} \rangle$, $\langle \mathbf{s} \otimes \mathbf{h} \rangle$, where $\mathbf{h}$ is the γ level recursive hash of $\mathbf{x}$,*

$$\Pr\left[\langle \mathbf{s}' \cdot x_j \rangle_E = \langle \mathbf{s}' \cdot x_j \rangle_G + \mathbf{s}' \cdot \mathbf{x}[j] \;\middle|\; \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \cdot x_j \rangle_G \leftarrow \mathsf{HRead}_G(\langle \mathbf{s} \otimes \mathbf{h} \rangle_G, j), \\ \langle \mathbf{s}' \cdot x_j \rangle_E \leftarrow \mathsf{HRead}_E(\langle \mathbf{s} \otimes \mathbf{h} \rangle_E, j, \mathbf{x}), \end{array}\right] > 1 - \mathsf{negl}(\lambda)$$

$$\Pr\left[\langle \mathbf{s}' \otimes \mathbf{h} \rangle_E = \langle \mathbf{s}' \otimes \mathbf{h} \rangle_G + \mathbf{s}' \otimes \mathbf{h} \;\middle|\; \begin{array}{c} \mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{s}'), \\ \langle \mathbf{s}' \otimes \mathbf{h} \rangle_G \leftarrow \mathsf{HCompress}_G(\langle \mathbf{s} \cdot \mathbf{x} \rangle_G), \\ \langle \mathbf{s}' \otimes \mathbf{h} \rangle_E \leftarrow \mathsf{HCompress}_E(\langle \mathbf{s} \cdot \mathbf{x} \rangle_E, \mathbf{x}), \end{array}\right] > 1 - \mathsf{negl}(\lambda).$$

Lemma 5.35 (Security of Hashing)**.** *Under the RLWE assumption, every efficient adversary $\mathcal{A}$ has negligible advantage in distinguishing the following experiments $\mathsf{Exp}_{\mathcal{A}}^b(1^\lambda)$, for $b \in \{0,1\}$*

- *Sample public parameters* pp*, and sends them to $\mathcal{A}$, who returns secret elements $\mathbf{s}' \in \mathcal{D}_{\mathsf{sk}}^2$.*
- *When $b = 0$, sample random exponents $\mathbf{s} \leftarrow \mathcal{D}_{\mathsf{sk}}^2$ and send* pd *to $\mathcal{A}$, computed as follows*

$$\mathsf{pd} \leftarrow \mathsf{FH.Setup}(1^\lambda, \mathbf{s}, \mathbf{s}').$$

- *When $b = 1$, sample* pd *with random $\mathcal{R}_q$ elements and send them to $\mathcal{A}$.*

Instantiation of HGate**.** Again, the sub-protocol HGate in our lattice instantiation is the same as the previous section (Figure 3), which runs the local algorithms EvalKey, EvalTag (Construction 5.24) from [ILL25b]. The only difference is the local algorithms EvalKey, EvalTag now use lattice based building blocks, MultKey, MultTag from Construction 5.36 as described below. The local algorithms EvalKey, EvalTag satsify correctness analogous to Lemma 5.25 and security analogous to Lemma 5.26 (under RLWE instead of DDH).

Construction 5.36 (aHMAC Multiplication under Lattices [ILL25c])**.** The construction is with respect to the same public parameters $\mathsf{pp} = (B, \mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ as specified in Construction 5.28.

$\mathsf{pd} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, s, s', s'' \in \mathcal{D}_{\mathsf{sk}})$**:** takes three secrets $s, s', s'' \in \mathcal{D}_{\mathsf{sk}}$, and computes pd as follows.

$$\begin{aligned} &a \leftarrow \mathcal{R}_q,\ e_1, e_2, e_3 \leftarrow \mathcal{D}_{\mathsf{err}},\ \alpha \leftarrow \mathcal{R}_q, \beta \leftarrow \mathcal{R}_p, \\ &b \leftarrow s \cdot a + e_1,\ b' \leftarrow s' \cdot a + e_2, \\ &c \leftarrow s' \cdot b + e_3 - s'' \cdot \Delta, \\ &\mathsf{pd} := (\mathsf{pp}, \alpha, \beta, a,\ b,\ b',\ c). \end{aligned}$$

$\langle s'' \cdot z \rangle_G \leftarrow \mathsf{MultKey}^{\mathsf{pd}}(\langle s \cdot x \rangle_G, \langle s' \cdot y \rangle_G)$**:** takes as inputs public data pd and two ring elements representing the 0-shares of some authenticated shares $\langle s \cdot x \rangle_G, \langle s' \cdot y \rangle_E \in \mathcal{R}$ w.r.t. some secrets $s, s' \in \mathcal{R}$. It outputs the 0-share of an authenticated share $\langle s'' \cdot z \rangle_G$.

$\langle s'' \cdot z \rangle_E \leftarrow \mathsf{MultTag}^{\mathsf{pd}}(\langle s \cdot x \rangle_E, \langle s' \cdot y \rangle_E, x, y)$**:** takes as input public data pd, the 1-shares of some authenticated shares $\langle s \cdot x \rangle_E, \langle s' \cdot y \rangle_E$ w.r.t. some secret s, s', and the shared values $x, y \in \mathcal{R}$. It outputs the 1-share of an authenticated share $\langle s'' \cdot z \rangle_E$.

Lemma 5.37 (Correctness of aHMAC Multiplication [ILL25c])**.** *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, public parameters* pp *as specified, secret elements* $s, s', s'' \in \mathcal{D}_{\mathsf{sk}}$*, inputs* $x, y \in \mathcal{R}_B$ *such that* $\|xy\|_\infty < B$*, and additive shares (over* $\mathcal{R}$*)* $\langle sx \rangle_G, \langle sx \rangle_E, \langle s'y \rangle_G, \langle s'y \rangle_E$*, the following holds:*

$$\Pr\left[\begin{array}{l|l} w_E = w_G + s'' \cdot xy & \mathsf{pd} \leftarrow \mathsf{Setup}(\mathsf{pp}, s, s', s'') \\ (\text{over } \mathcal{R}) & w_G = \mathsf{MultKey}(\langle sx \rangle_G, \langle s'y \rangle_G) \\ & w_E = \mathsf{MultTag}(\langle sx \rangle_E, \langle s'y \rangle_E, x, y) \end{array}\right] > 1 - \mathsf{negl}(\lambda).$$

Instantiation of Setup **and** Enc**.** We now describe the $\mathsf{Setup}(1^\lambda, 1^D, |C|)$ procedure that selects public parameters $\mathsf{pp} := (1^D, \mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}}, \mathsf{Hash}, \gamma = \log_\lambda |C|)$ for all sub-protocols HRead, HCompress, HGate.

- Let $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ be a polynomial ring, where $n(\lambda) < \mathsf{poly}(\lambda)$ is a power-of-two, and $\mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}}$ be truncated discrete gaussian distribution over $\mathcal{R}$ with coefficients bounded by $\mathsf{poly}(\lambda)$.
- Select moduli p, q satisfying requirements from Construction 5.28, under $B = \lambda^{\omega(1)} > (n \cdot \lambda \cdot \lceil \log q \rceil)^\gamma$.
- Let $\mathcal{D}_{\mathsf{pub}}$ be as specified in Construction 5.30. Sample $b_i \leftarrow \mathcal{D}_{\mathsf{pub}}$ for $i \in \lambda \cdot \log q$, which define a global hash function $\mathsf{Hash} : \mathcal{R}^{\lambda \cdot \lceil \log q \rceil} \to \mathcal{R}^{\lceil \log q \rceil}$ with compression factor λ as specified in Equation 5.6.

The sampling of gk, and the $\mathsf{Enc}(\mathsf{gk}, \mathbf{x})$ procedure that produces encodings $[\![\mathbf{x}]\!]$, public data pd is analogous to the previous section (Equation 5.3, 5.4, and 5.5), except the secrets are sampled from $\mathcal{D}_{\mathsf{sk}}^2$ instead of $[p]^2$.

Garbler Depth Complexity. Similarly to the group instantiation, we review the depth of each of the sub-protocols HRead, HCompress, HGate.

HRead**:** Mainly consists of γ sequential calls to Extract.

Extract consists mainly of a ring multiplication over $\mathcal{R}_q$ and a coefficient rounding. Both of these steps have depth $O(\log \lambda)$.

We conclude the depth of HRead is $O(\gamma \cdot \log \lambda)$, where $\gamma = \log_\lambda |C|$.

HCompress**:** Mainly consists of a call to SecSwitch, and γ sequential calls to Compress.

SecSwitch similarly consists of two ring multiplications and a coefficient rounding, and has depth $O(\log \lambda)$. Compress consists of a ring multiplication and a summation over λ ring elements. Both of these steps have depth $O(\log \lambda)$.

We conclude the depth of HCompress is also $O(\gamma \cdot \log \lambda)$.

HGate**:** Mainly consists of a call to SecSwitch and MultKey. The construction of MultKey from [ILL25b] consists of constant number of ring multiplications and a coefficient rounding, similarly to our SecSwitch construction, and hence has depth $O(\log \lambda)$.

Succinctness and Decomposability. Similarly to the group instantiation, the garbling material consists of the public data pd and a series of re-key materials $\{\mathsf{rk}^{(d)}\}_{d \in [D]}$. Thanks to recursive hashing, the public data and re-key material for each depth $d \in [D]$ have a fixed size $\mathsf{poly}(\lambda)$, independent of $|C|$. Overall, they have size $< D \cdot \mathsf{poly}(\lambda)$.

The analysis for decomposability and the size of input encodings and dk are analogous to the group instantiation.

Security. The selective one-wayness can be shown analogously to that of the group instantiation, based on the standard RLWE assumption (Definition 5.12).

6 Non-Reuseable Partially Succinct Garbling

In this section, we will show the *private* analogue of the constructions in Section 5.

Theorem 6.1 (Non-Reuseable Partially Succinct Garbling under DDH)**.** *Assuming any of the following:*

- *DDH in prime-order groups that are subgroups of $\mathbb{Z}_p^*$ for some prime p;*
- *DDH in Paillier groups;*

And additionally a PRF in NC1, there exists a succinct garbling scheme satsifying the following.

- *Garbling sizes of a circuit $C : \{0,1\}^\ell \to \{0,1\}$ with depth D satisfy:*

$$|\hat{C}| < (D + |C|^{2/3} \cdot D^{1/3}) \cdot \mathsf{poly}(\lambda) + |C|,\ |\hat{x}| < \ell \cdot \mathsf{poly}(\lambda),\ |\mathsf{dk}| < \mathsf{poly}(\lambda).$$

- $\mathsf{GarbC}(1^\lambda, C)$ *can be computed in depth* $\mathsf{poly}(\lambda, \log |C|)$.

Further assuming a local PRG with constant depth, then $\mathsf{GarbC}(1^\lambda, C)$ *can be computed in NC1. The instantiation under DDH in prime-oder groups suffers a* $1/\mathsf{poly}(\lambda)$ *correctness and privacy error, which can be removed via privacy amplification at the cost of increasing the depth of* $\mathsf{GarbC}(1^\lambda, C)$ *to* $\mathsf{poly}(\lambda, \log C)$ *again.*

Theorem 6.2 (Non-Reuseable Partially Succinct Garbling under RLWE)**.** *Assuming RLWE with respect to* $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ *as specified in Construction 5.28 under $B = 2$, and additionally a PRF in NC1, there exists a succinct garbling scheme satsifying the following.*

- *Garbling sizes of a circuit $C : \{0,1\}^\ell \to \{0,1\}$ with depth D satisfy:*

$$|\hat{C}| < D \cdot \mathsf{poly}(\lambda) + |C|,\ |\hat{x}| < \ell \cdot \mathsf{poly}(\lambda),\ |\mathsf{dk}| < \mathsf{poly}(\lambda).$$

- $\mathsf{GarbC}(1^\lambda, C)$ *can be computed in depth* $\mathsf{poly}(\lambda, \log |C|)$.

Further assuming a local PRG with constant depth, then $\mathsf{GarbC}(1^\lambda, C)$ *can be computed in NC1.*

6.1 Instantiation Under Groups

The starting point of our group instantiation is the (leveled) partially succinct garbling scheme of [ILL25b]. In particular, it contains a sub-protocol that allows two parties P_G, P_E who jointly hold additive shares of the form $\langle \mathbf{s} \cdot \overline{x} \rangle$, $\langle \mathbf{s} \cdot \overline{y} \rangle$, to derive additive shares $\langle \mathbf{s}' \cdot \overline{g(x,y)} \rangle$, for any Boolean gate g, where the notation $\overline{x} := x \oplus r_x$ denotes a masked value x by a pseudo-random mask r_x. We will directly use this sub-protocol as our HGate protocol. Together with the HRead, HCompress sub-protocols from Section 5, we obtain a partially succinct garbling scheme (with privacy).

In more detail, the encoding of a wire value $[\![x]\!]$ for $x \in \{0,1\}$ is the same as Section 5.2, except the wire value x is masked as $\overline{x}$. The Rekey steps are exactly the same as Section 5.2.

$$[\![x]\!] = ([\![x]\!]_G, [\![x]\!]_E),\ \text{where } [\![x]\!]_G = (\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{x} \rangle_G, \mathsf{id}),\ \text{and } [\![x]\!]_E = (\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{x} \rangle_E, \mathsf{id}, \overline{x}).$$

In the following, we first recall necessary preliminaries for the HGate sub-protocol adapted from [ILL25a], and then explain the Setup and Enc algorithms for our garbling scheme. (The HRead, HCompress sub-protocols remain the same as Section 5.2.)

Protocol HGate

The protocol runs between a garbler P_G and an evaluator P_E. It uses the following ingredients:

- The algorithms EvalKey, EvalTag from Construction 5.24.
- The algorithms ExtEval from Construction 6.3.
- A $\mathsf{PRF} : \{0,1\}^\lambda \times [2^\lambda] \to \{0,1\}$ in NC1.

Inputs: P_G, P_E hold public data pd, an operator op, and jointly encodings of $[\![x]\!]$, $[\![y]\!]$, (with $\mathsf{id} = (d, \mathsf{InWire})$). We assume op also contains the input and output wire ids w_1, w_2, w_3.
Outputs: P_G, P_E jointly hold encodings $[\![z]\!]$ (with $\mathsf{id}' = (d, \mathsf{OutWire})$), where z should equal $\mathsf{op}(x, y)$.

- P_G reads $\mathsf{GE.pd}^{(d)}$ from pd, $\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{x} \rangle_G$ $\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{y} \rangle_G$ from $[\![x]\!]_G, [\![x]\!]_G$, and runs

$$\langle \mathbf{s}^{\mathsf{id}''} \cdot \mathbf{u}[i] \rangle_G \leftarrow \mathsf{EvalKey}^{\mathsf{GE.pd}^{(d)}}(\mathsf{eq}_i, \langle \mathbf{s}^{\mathsf{id}} \cdot \overline{x} \rangle_G, \langle \mathbf{s}^{\mathsf{id}} \cdot \overline{y} \rangle_G) \text{ for } i \in [4],$$

  where $\mathsf{id}'' = (d, \mathsf{TmpWire})$, indicating an intermediate secret $\mathbf{s}^{\mathsf{id}''} \in \{0,1\}^{\lceil 3 \cdot \log p \rceil}$, and where eq_i is a binary gate checking if $(\overline{x}, \overline{y})$ represents i. Then, it reads $\mathsf{HSS.pd}_{\mathsf{key}}^{(d)}$ from pd, and runs

$$\langle \mathbf{z} \rangle_G \leftarrow \mathsf{ExtEval}^{\mathsf{HSS.pd}_{\mathsf{key}}^{(d)}}(f_{\mathsf{op}}, \{\langle \mathbf{s}^{\mathsf{id}''} \cdot \mathbf{u}[i] \rangle_G, \langle \mathbf{u}[i] \rangle_G\}_{i \in [4]}),$$

  where $\langle \mathbf{u}[i] \rangle_G \leftarrow 0$, and f_{op} is defined as follows,

$$f_{\mathsf{op}}(\mathsf{key} \| \mathsf{Bits}(\mathbf{s}^{\mathsf{id}'})) = \mathsf{Bits}(1, \mathbf{s}^{\mathsf{id}'}) \otimes \mathsf{Tb},$$

  and $\mathsf{Tb} \in \{0,1\}^{1 \times 4}$ is a truth table of op, adjusted for masks over the inputs: [a]

$$\begin{aligned} \mathsf{Tb}[(\overline{x}, \overline{y})] &= \mathsf{op}(\overline{x} \oplus r_x, \overline{y} \oplus r_y) \oplus r_z, \text{ where} \\ & r_x \leftarrow \mathsf{PRF}(\mathsf{key}, w_1),\ r_y \leftarrow \mathsf{PRF}(\mathsf{key}, w_2),\ r_z \leftarrow \mathsf{PRF}(\mathsf{key}, w_3). \end{aligned} \tag{6.1}$$

  It sets $\langle \overline{z} \rangle_G \leftarrow \langle \mathbf{z}[0] \rangle_G$, and $\langle \mathsf{Bits}(\mathbf{s}^{\mathsf{id}'}) \cdot \overline{z} \rangle_G \leftarrow \langle \mathbf{z}[1:] \rangle_G$.

- P_E analogously runs EvalTag and ExtEval with $\langle \mathbf{u}[i] \rangle_E \leftarrow \mathbf{u}[i]$ to derive $\langle \overline{z} \rangle_E$ and $\langle \mathbf{s}^{\mathsf{id}'} \cdot \overline{z} \rangle_E$.
- P_G sends its share $\langle \overline{z} \rangle_G$ mod 2 to P_E, who recovers $\overline{z}$. They respectively output $[\![z]\!]_G := (\langle \mathbf{s}^{\mathsf{id}'} \cdot \overline{z} \rangle_G, \mathsf{id}')$, and $[\![z]\!]_E := (\langle \mathbf{s}^{\mathsf{id}'} \cdot \overline{z} \rangle_E, \mathsf{id}', \overline{z})$.

[a] When the output of op is an output of the circuit, we avoid the wire mask by setting $r_z = 0$.

Figure 4: The HGate sub-protocol with privacy.

Instantiation of HGate**.** We recall necessary preliminaries on the (leveled) construction of HGate (Figure 4) based on DDH from [ILL25b]. The sub-protocol consists first of local computations by both parties, running the EvalKey, EvalTag algorithms from Construction 5.24 and then the ExtEval algorithm which we describe below in Construction 6.3. In the end, the protocol incurs 1-bit communication from the garbler P_G to the evaluator P_E. In our description (Figure 4), the sub-protocol assume the parties P_G, P_E hold public data pd produced in the overall protocol by Enc.

Construction 6.3 (Extended HSS Evaluation under DDH [BGI16])**.** The construction is with respect to the same public parameters (B, δ, G, p, g) as Construction 5.13 under $B = 2$, and additionally

- $B_f < \mathsf{poly}(\lambda)$ is a polynomial circuit size bound.

It uses as ingredients the DDLog algorithm from Lemma 5.5 instantiated with a suitable $\mathsf{PRF} : \{0,1\}^\lambda \times G \to \{0,1\}^\lambda$

to achieve error probability $< \delta/\mathsf{poly}(B_f)$ (s.t. the error probability over all invocations of DDLog is $< \delta$).

$\mathsf{pd}_{\mathbf{m}} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, \mathbf{s} \in \{0,1\}^{\lceil 3\log p\rceil}, \mathbf{m} \in \{0,1\}^{\ell_m})$**:** First sample random coefficients $\mathbf{c} \leftarrow [p]^{\lceil 3\log p\rceil}$, and define $s = \mathbf{c}^\intercal \mathbf{s} \bmod p$. Then compute public data $\mathsf{pd}_{\mathbf{m}}$ w.r.t. $\mathbf{m}$ as follows:

$$\mathbf{r} \leftarrow \mathbb{Z}_p^{\ell_m},\ \mathbf{R} \leftarrow \mathbb{Z}_p^{\ell_m \times \lceil 3\log p\rceil},$$
$$\mathsf{pd}_{\mathbf{m}} := (\mathsf{pp}, g^{\mathbf{c}\otimes\mathbf{r}}, g^{\mathbf{c}\otimes\mathbf{R}}, g^{s\cdot\mathbf{r}+\mathbf{m}}, g^{s\cdot\mathbf{R}+\mathbf{m}\otimes\mathbf{s}}).$$

$\langle \mathbf{z}\rangle_{G/E} \leftarrow \mathsf{ExtEval}^{\mathsf{pd}_{\mathbf{m}}}(f : \{0,1\}^{\ell_m} \to \{0,1\}^{\ell_z \times \ell_x}, \{\langle \mathbf{s}\cdot x_i\rangle_{G/E}, \langle x_i\rangle_{G/E}\}_{i\in[\ell_x]})$**:** Takes an NC1 Boolean circuit f and ℓ_x shares $\langle \mathbf{s}\cdot x_i\rangle_{G/E}$, $\langle x_i\rangle_{G/E}$, w.r.t. the secret $\mathbf{s} \in \{0,1\}^{\lceil 3\log p\rceil}$ as input. Outputs an additive share $\langle \mathbf{z}\rangle_{G/E}$ where $\mathbf{z} \in \mathbb{Z}^{\ell_z}$ should equal

$$\mathbf{z} = f(\mathbf{m})\cdot\mathbf{x}, \text{ where } \mathbf{x} := (\ldots x_i \ldots), \text{ and the matrix product is over } \mathbb{Z}.$$

Lemma 6.4 (Correctness of Extended HSS Evaluation [BGI16])**.** *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, public parameters* pp *as specified, secret string* $\mathbf{s} \in \{0,1\}^{\lceil 3\log p\rceil}$*, NC1 Boolean circuit* $f : \{0,1\}^{\ell_m} \to \{0,1\}^{\ell_x}$ *with size* $|f| \le B_f$*, strings* $\mathbf{m} \in \{0,1\}^{\ell_m}$, $\mathbf{x} \in \{0,1\}^{\ell_x}$*, and additive shares* $\langle \mathbf{s}\otimes\mathbf{x}\rangle_G$, $\langle \mathbf{s}\otimes\mathbf{x}\rangle_E$*, following holds:*

$$\Pr\left[\mathbf{w}_E = \mathbf{w}_G + f(\mathbf{m})\cdot\mathbf{x} \ (\text{over } \mathbb{Z}_p) \;\middle|\; \begin{array}{l}\mathsf{pd}_{\mathbf{m}} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{m})\\ \mathbf{w}_G := \mathsf{ExtEval}(f, \{\langle \mathbf{s}\otimes\mathbf{x}[i]\rangle_G, \langle \mathbf{x}[i]\rangle_G\}),\\ \mathbf{w}_E := \mathsf{ExtEval}(f, \{\langle \mathbf{s}\otimes\mathbf{x}[i]\rangle_E, \langle \mathbf{x}[i]\rangle_E\})\end{array}\right] > 1 - \mathsf{negl}(\lambda) - \delta(\lambda).$$

Remark 6.5. We note a mismatch between our encoding format, which requires a secret vector $\mathbf{s} \in [p]^2$, and the extended HSS evaluation interface, which requires a secret string $\mathbf{s} \in \{0,1\}^{\lceil 3\log p\rceil}$. We resolve this issue by introducing an intermediate encoding format, indicated by TmpWire, using a string as the secret as required by the HSS evaluation interface. Such intermediate encodings will only exist as outputs of the EvalKey, EvalTag algorithms (Construction 5.24), and as inputs to the ExtEval algorithms. [12] (See our HGate sub-protocol, Figure 4, and our instantiation of the overall Setup algorithm, Equation 6.2).

For our security proof later, we note here that the public data pd produced by the Setup of extended HSS evaluations under a random secret $\mathbf{s} \in \{0,1\}^{\lceil 3\log p\rceil}$ are computationally indistinguishable from random. The following lemma follows directly from the security of the BHHO [BHHO08] crypto-system. We refer to [BHHO08] for details.

Lemma 6.6 (Security of HSS Evaluation)**.** *Under the DDH assumption, every efficient adversary* $\mathcal{A}$ *has negligible advantage in distinguishing the following experiments* $\mathsf{Exp}^b_{\mathcal{A}}(1^\lambda)$*, for* $b \in \{0,1\}$

- *Sample public parameters* pp*, and sends them to* $\mathcal{A}$*, who returns bits* $\mathbf{m} \in \{0,1\}^{\ell_m}$*.*
- *When* $b = 0$*, sample random exponents* $\mathbf{s} \leftarrow \{0,1\}^{\lceil 3\log p\rceil}$ *and send* pd *to* $\mathcal{A}$*, computed as follows*

$$\mathsf{pd} \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{s}, \mathbf{m}),$$

- *When* $b = 1$*, sample* pd*, with random group elements from* G *and send them to* $\mathcal{A}$*.*

Instantiation of Setup **and** Enc**.** The $\mathsf{Setup}(1^\lambda, 1^D, |C|)$ procedure that selects public parameters $\mathsf{pp} := (1^D, G, p, g, \mathsf{Hash}, \delta, \gamma = \lceil |C|^{2/3}\cdot D^{1/3}\rceil, B_f)$ for all sub-protocols HRead, HCompress, HGate is the same as described in Section 5.2, except additionally set B_f as follows:

- Set B_f to be the circuit size computing masked truth tables as described in Equation 6.1. We have $B_f \le O(|\mathsf{PRF}|)$

We next describe how to sample gk and the $\mathsf{Enc}(\mathsf{gk}, \mathbf{x})$ procedure that produces encodings $[\![\mathbf{x}]\!]$ and public data pd.

[12] EvalKey, EvalTag can be modified in the natural way to take $\mathbf{s}' \in \{0,1\}^{\lceil 3\log p\rceil}$ during their setup, and output such intermediate encodings.

- First choose leveled secrets $\mathbf{s}^{\mathsf{id}} \in [p]^2$ as gk for our encodings as in Equation 5.3, and additionally the following

$$\mathbf{s}^{(d,\mathsf{TmpWire})} \leftarrow \{0,1\}^{\lceil 3 \log p \rceil} \text{ for } d = 1, \ldots, D.$$

- Next set up public data for the sub-protocols as in Equation 5.4, except sampling $\mathsf{GE.pd}^{(d)}$ differently and $\mathsf{HSS.pd}^{(d)}$ additionally as follows:

$$\begin{gathered} \mathsf{GE.pd}^{(d)} \leftarrow \mathsf{GE.Setup}(1^\lambda, \mathbf{s}^{\mathsf{id}}, \mathbf{s}^{\mathsf{id}''}),\ \mathsf{HSS.pd}^{(d)}_{\mathsf{key}} \leftarrow \mathsf{HSS.Setup}(1^\lambda, \mathbf{s}^{\mathsf{id}''}, (\mathsf{key} \| \mathsf{Bits}(\mathbf{s}^{\mathsf{id}'}))) \\ \text{where } \mathsf{id} = (d, \mathsf{InWire}),\ \mathsf{id}'' = (d, \mathsf{TmpType}),\ \mathsf{id}' = (d, \mathsf{OutWire}),\ \mathsf{key} \leftarrow \{0,1\}^\lambda,\ \text{for } d = 1, \ldots, D. \end{gathered} \tag{6.2}$$

- Finally use $\mathbf{s}^{\mathsf{id}=(0,\mathsf{OutWire})}$ from the above to compute

$$\begin{gathered} \overline{\mathbf{x}}[i] \leftarrow \mathbf{x}[i] \oplus \mathsf{PRF}(\mathsf{key}, i),\ \langle \mathbf{s}^{\mathsf{id}} \cdot \overline{\mathbf{x}}[i] \rangle_G \leftarrow [p]^2, \\ \langle \mathbf{s}^{\mathsf{id}} \cdot \mathbf{x}[i] \rangle_E \leftarrow \langle \mathbf{s}^{\mathsf{id}} \cdot \overline{\mathbf{x}}[i] \rangle_G + \mathbf{s}^{\mathsf{id}} \cdot \overline{\mathbf{x}}[i] \text{ over } \mathbb{Z}_p, \\ [\![\mathbf{x}[i]]\!]_G := (\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{\mathbf{x}}[i] \rangle_G, \mathsf{id}), \quad [\![\mathbf{x}[i]]\!]_E := (\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{\mathbf{x}}[i] \rangle_E, \mathsf{id}, \overline{\mathbf{x}}[i]). \end{gathered} \tag{6.3}$$

Garbler Depth Complexity. Compared to the privacy-free construction in Section 5.2 only HGate changes.

HGate**:** The garbler first runs EvalKey, which has depth $O(\log \lambda)$. It then runs ExtEval to evaluate the function f_{op} as described in Equation 6.1, which has depth $\mathsf{poly}(\lambda)$ in general. Hence HGate has depth $\mathsf{poly}(\lambda)$.

We note that when replacing PRF with a local PRG with constant depth, then the depth of ExtEval and hence of HGate becomes $O(\log \lambda)$. The size of key for the local PRG needs to become larger as a trade-off.

Succinctness and Decomposability. The garbling material consists of the public data and re-key material of size $< (D + |C|^{2/3} \cdot D^{1/3}) \cdot \mathsf{poly}(\lambda)$, similarly to the privacy-free construction in Section 5.2, and additionally of 1-bit per gate from the communication in HGate sub-protocol. Therefore, the overall garbling material have size $< |C| + (D + |C|^{2/3} \cdot D^{1/3}) \cdot \mathsf{poly}(\lambda)$.

The input encoding and decryption key have the same sizes as the privacy-free construction in Section 5.2.

Removing the $1/\mathsf{poly}(\lambda)$ Correctness and Privacy Error. Similarly to the privacy-free construction in Section 5.2, we obtain a garbling scheme with 1/poly correctness and privacy error. The two solutions described there (1) using Paillier groups instead of prime-order groups and (2) applying privacy amplification techniques from [ILL25c, BGI17] can also be applied here. We note that even when assuming a local PRG with constant depth, the privacy amplification from solution (2) will make the depth of HGate become $\mathsf{poly}(\lambda)$.

Security. We sketch *selective* semantic security (Definition 3.2, with privacy error $2 \cdot \delta(\lambda) = 1/\mathsf{poly}(\lambda)$) for the garbling scheme via a series of hybrid experiments, where Hyb_0 is the experiment $\mathsf{ExptIND}^{(0)}_{\mathcal{A}}(\lambda)$, and Hyb_3 is the experiment $\mathsf{ExptIND}^{(1)}_{\mathcal{A}}(\lambda)$.

- Hyb_0: We briefly recall the experiment $\mathsf{ExptIND}^{(0)}_{\mathcal{A}}(\lambda)$:
  - The challenger returns public parameters pp according to Setup, and receives a circuit C and input $\mathbf{x}_0$ from the adversary.
  - The challenger returns $\widehat{C}$, which consists of the public data pd as described in Equation 5.4 and Equation 6.2, rekey materials $\{\mathsf{rk}^{(d)}\}$, and a bit $b^{(i)}$ for every intermediate wire i.
  - The challenger also returns input encodings $[\![\mathbf{x}]\!]_E$ as described in Equation 6.3.
- Hyb_1: This hybrid computes the rekey materials $\{\mathsf{rk}^{(d)}\}$ and the bits $b^{(i)}$ differently.
  - Using the input encodings $[\![\mathbf{x}]\!]_E$, derive evaluator's shares on all intermediate wires and hashes, and set $\{\mathsf{rk}^{(d)}\}$ as the difference between evaluators shares of hashes, instead of the garbler's.

– Derive the bits $b^{(i)}$ as evaluator's share of $\langle \overline{z} \rangle_E$ on the wire i, subtracting the value $\overline{z}$, which can be derived from the correct wire value z on wire i and the pseudo-random mask r_z on it.

By the correctness of the garbling scheme, this hybrid is equivalent to the previous, except with $\delta(\lambda) = 1/\mathsf{poly}(\lambda)$ failure probability.

$$|\Pr[\mathsf{Hyb}_{1,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}_{0,\mathcal{A}}(\lambda) = 1]| \leq \delta(\lambda).$$

- Hyb_2: This hybrid computes the input encodings as random elements in $\mathbb{Z}_q^2$, instead of following Equation 6.3. This hybrid is equivalent to the previous, by the randomness of $\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{\mathbf{x}}[i] \rangle_G$. (See Equation 6.3).

$$\Pr[\mathsf{Hyb}_{2,\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}_{1,\mathcal{A}}(\lambda) = 1].$$

We note that the secret $\mathbf{s}^{(0,\mathsf{OutWire})}$ is only used for generatring public data $\mathsf{HW.pd}^{(0)}$, and not anywhere else in the experiment.

- This hybrid samples the public data $\{\mathsf{HW.pd}^{(d)}\}$, $\{\mathsf{HR.pd}^{(d)}\}$, $\{\mathsf{GE.pd}^{(d)}\}$, $\{\mathsf{HSS.pd}^{(d)}_{\mathsf{key}}\}$ as random group elements in G, instead of a chain of ciphertexts encrypting the next secret s' under some secret s. By Lemma 5.21, 5.26, and 6.6, all of which assume the DDH assumption in prime-order groups, we have

$$|\Pr[\mathsf{Hyb}_{3,\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}_{2,\mathcal{A}}(\lambda) = 1]| < \mathsf{negl}(\lambda).$$

We note that the PRF key no longer used anywhere in the experiment, except for generating the wire masks for every intermediate wire.

- Hyb_4: This hybrid samples the masked wire values as random bits, in particular, the $\overline{z}$ values used for generating bits $b^{(i)}$ as described in Hyb_1 are now random bits, independent of the input $\mathbf{x}_0$. By the security of PRF, we have

$$|\Pr[\mathsf{Hyb}_{4,\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}_{3,\mathcal{A}}(\lambda) = 1]| < \mathsf{negl}(\lambda).$$

- Hyb_5: This is the experiment $\mathsf{ExptIND}^{(1)}_{\mathcal{A}}(\lambda)$. By a symmetric arguments as above, we have

$$|\Pr[\mathsf{Hyb}_{5,\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}_{4,\mathcal{A}}(\lambda) = 1]| < \mathsf{negl}(\lambda) + \delta(\lambda).$$

By a standard hybrid argument, we have

$$|\Pr[\mathsf{ExptIND}^{(1)}_{\mathcal{A}}(\lambda)] - \mathsf{ExptIND}^{(0)}_{\mathcal{A}}(\lambda)| < \mathsf{negl}(\lambda) + 2\delta(\lambda).$$

6.2 Instantiation Under Lattices

Similarly to the group instantiation, which combines the group constructions of HRead, HCompress from Section 5.2 and the group construction of HGate from [ILL25b], we can instead combine the lattice constructions of HRead, HCompress from Section 5.3 and the lattice construction of HGate from [ILL25b] to obtain a low-depth partially succinct garbling with privacy.

In more detail, the encoding $[\![x]\!]$ of wire values x now consists of additive shares over $\mathcal{R}_p$ and masked wire values in the clear:

$$[\![x]\!] = ([\![x]\!]_G, [\![x]\!]_E), \text{ where } [\![x]\!]_G = (\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{x} \rangle_G, \mathsf{id}), \text{ and } [\![x]\!]_E = (\langle \mathbf{s}^{\mathsf{id}} \cdot \overline{x} \rangle_E, \mathsf{id}, \overline{x}).$$

The Rekey steps, HRead, HCompress protocols are the same as in Section 5.3.

Instantiation of HGate**.** The HGate sub-protocol stays the same as (Figure 4), except using the lattice constructions of the ingredients, EvalKey, EvalTag from Section 5.3 and ExtEval which we describe below in Construction 6.7.

Construction 6.7 (Extended HSS Evaluation under lattice [BKS19])**.** The construction is with respect to the same public parameters $(\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ as Construction 5.28 under $B = 2$. It uses as ingredients: $\mathsf{PRF} : \{0,1\}^\lambda \times [2^\lambda] \to \mathcal{R}_p$. $\mathsf{PRF}' : \{0,1\}^\lambda \times [2^\lambda] \to \mathcal{R}_q$.

$\mathsf{pd}_{\mathbf{m}} \leftarrow \mathsf{Setup}^{\mathsf{pp}}(1^\lambda, s \in \mathcal{D}_{\mathsf{sk}}, \mathbf{m} \in \{0,1\}^{\ell_m})$**:** Compute public data $\mathsf{pd}_{\mathbf{m}}$ w.r.t. $\mathbf{m}$ as follows:

$$\begin{aligned} &\mathbf{a} \leftarrow \mathcal{R}_q^{\ell_m},\ r_1, r_2 \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_1', \mathbf{e}_2' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_m},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda, \\ &\mathbf{b} := s \cdot \mathbf{a} + \mathbf{e}, \quad \mathbf{c}_1 := r_1 \cdot \mathbf{a} + \mathbf{e}_1 + \mathbf{m} \cdot \Delta, \quad \mathbf{c}_1' := r_1 \cdot \mathbf{b} + \mathbf{e}_1', \\ &\qquad\qquad \mathbf{c}_2 := r_2 \cdot \mathbf{a} + \mathbf{e}_2, \qquad\qquad \mathbf{c}_2' := r_2 \cdot \mathbf{b} + \mathbf{e}_2' + \mathbf{m} \cdot \Delta. \\ &\mathsf{pd}_{\mathbf{m}} := (\mathsf{pp}, \mathsf{seed}, \mathbf{c}_1, \mathbf{c}_1', \mathbf{c}_2, \mathbf{c}_2'). \end{aligned}$$

$\langle \mathbf{z} \rangle_{G/E} \leftarrow \mathsf{ExtEval}^{\mathsf{pd}_{\mathbf{m}}}(f : \{0,1\}^{\ell_m} \to \{0,1\}^{\ell_z \times \ell_x}, \{\langle s \cdot x_i \rangle_{G/E}, \langle x_i \rangle_{G/E}\}_{i \in [\ell_x]})$**:** Takes an NC1 Boolean circuit f and ℓ_x shares $\langle s \cdot x_i \rangle_{G/E}$, $\langle x_i \rangle_{G/E}$, w.r.t. the global secret $s \in \mathcal{D}_{\mathsf{sk}}$ as input. Outputs an additive share $\langle \mathbf{z} \rangle_{G/E}$ where $\mathbf{z}$ should equal

$$\mathbf{z} = f(\mathbf{m}) \cdot \mathbf{x} \text{ over } \mathbb{Z}.$$

Lemma 6.8 (Correctness of Extended HSS Evaluation [BKS19])**.** *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, public parameters* pp *as specified, secret elements* $s \in \mathcal{D}_{\mathsf{sk}}$*, NC1 Boolean circuit* $f : \{0,1\}^{\ell_m} \to \{0,1\}^{\ell_z \times \ell_x}$ *strings* $\mathbf{m} \in \{0,1\}^{\ell_m}$, $\mathbf{x} \in \{0,1\}^{\ell_x}$*, and additive shares* $\langle s \cdot \mathbf{x} \rangle_G$, $\langle s \cdot \mathbf{x} \rangle_E$, $\langle \mathbf{x} \rangle_G$, $\langle \mathbf{x} \rangle_E$*, following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_E = \mathbf{w}_G + f(\mathbf{m})^\intercal \mathbf{x} & \mathsf{pd}_{\mathbf{m}} \leftarrow \mathsf{Setup}(1^\lambda, s, \mathbf{m}) \\ \quad (\text{over } \mathcal{R}_p) & \mathbf{w}_G = \mathsf{ExtEval}(f, \{\langle s \cdot \mathbf{x}[i] \rangle_G, \langle \mathbf{x}[i] \rangle_G\}), \\ & \mathbf{w}_E = \mathsf{ExtEval}(f, \{\langle s \cdot \mathbf{x}[i] \rangle_E, \langle \mathbf{x}[i] \rangle_E\}) \end{array}\right] > 1 - \mathsf{negl}(\lambda).$$

Remark 6.9. The mismatch between our encoding format, which requires a secret vector $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^2$, and the extended HSS evaluation interface, which requires a secret $s \in \mathcal{D}_{\mathsf{sk}}$ can be resolved analogously to the group instantiation (Section 6.1) by introducing an intermediate encoding format, indicated by TmpWire, using a single element as the secret as required by the HSS evaluation interface.

Similarly to the group instantiation (Section 6.1), we note the following straightforward lemma and omit its proof.

Lemma 6.10 (Security of HSS Evaluation)**.** *Under the RLWE assumption, every efficient adversary* $\mathcal{A}$ *has negligible advantage in distinguishing the experiments* $\mathsf{Exp}_{\mathcal{A}}^b(1^\lambda)$*, for* $b \in \{0,1\}$*, defined analogously to Lemma 6.6.*

Instantiation of Setup **and** Enc**.** The $\mathsf{Setup}(1^\lambda, 1^D, |C|)$ and Enc steps are the same as Section 5.3, with analogous modifications to group construction in Section 6.1.

Garbler Depth Complexity. Similarly to the group construction in Section 6.1, the depth of HGate is $\mathsf{poly}(\lambda)$ in general. When replacing PRF with a local PRG with constant depth, the depth of HGate becomes $O(\log \lambda)$.

Succinctness and Decomposability. Similarly to the group construction in Section 6.1, the overall garbling material have size $< |C| + D \cdot \mathsf{poly}(\lambda)$.

The input encoding and decryption key have the same sizes as the privacy-free construction in Section 5.3.

Security. The selective semantice security can be shown analogously to that of the group instantiation, based on the standard RLWE assumption (Definition 5.12).

7 Reusable Succinct Privacy-Free Garbling

In this section, we prove the following statement:

Theorem 7.1 (Reusable Succinct Privacy-Free Garbling)**.** *Assuming decomposed LWE Assumption 7.5 (parameterized as Construction 7.12), there exists a reusable succinct privacy-free garbling scheme* Π_{garble} *satisfying the following:*

- *For a circuit* $C : \{0,1\}^\ell \to \{0,1\}$ *of depth* d*, the public data* pd *and garbling material* $|\hat{C}|, |\hat{x}|, |\mathsf{gk}|$ *are of size* $\mathsf{poly}(\lambda, d)$ *(independent of circuit size* $|C|$*). The garbled input* $\hat{x}$ *additionally on* ℓ *and is decomposable.*
- *The garbling algorithms* $(\mathsf{Setup}, \mathsf{GarbC}, \mathsf{GarbInp})$ *are computable in* NC^1*.*
- Π_{garble} *satisfies selective authenticity.*

7.1 Preliminaries

Notation. For a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m}$, we use $||\mathbf{M}||$ to denote the maximum absolute value of its entries. We use $\mathbf{A} \otimes \mathbf{B}$ to denote the Kronecker product between $\mathbf{A}$ and $\mathbf{B}$. We will use $\mathbf{I}$ to denote the identity matrix. We reserve the symbol $\mathbf{G}$ to denote the $n \times m$ gadget matrix $\mathbf{I}_n \otimes [1, 2, 4, \ldots 2^{\lceil \log q \rceil} - 1, 0, ..., 0]$. We let $\mathbf{G}^{-1}(\cdot) : \mathbb{Z}_q^n \to \mathbb{Z}_q^m$ denote entry-wise bit decomposition (padded with zeroes when necessary) where $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$. We use $\mathcal{D}_\sigma$ to refer to the *uniform* distribution over range $[-\sigma, \sigma] \subset \mathbb{Z}$[13]. We use $\mathcal{D} \approx_s \mathcal{D}'$ to denote distributions with negligible (in security parameter λ) statistical distance. We use $a(\lambda) \gg b(\lambda)$ to denote a superpolynomial gap between $a(\lambda)$ and $b(\lambda)$ - i.e. $a(\lambda) \geq b(\lambda) \cdot \lambda^{\omega(1)}$. We use $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ to denote the rounding operation, where we interpret elements of $\mathbb{Z}_q$ as elements of $\mathbb{Z}$ and multiply by (q/p) before rounding to the nearest integer. We will use $\lfloor \cdot \rceil_p$.

Lemma 7.2 (Smudging Lemma (Adapted From [AJL+12]))**.** *Let* λ *be a security parameter and* $B = B(\lambda), B' = B'(\lambda)$ *be functions where* $B \ll B'$*. Then the distributions*

$$\mathcal{D}_{B+B'} \approx_s \mathcal{D}_{B'}$$

are statistically close.

Lemma 7.3 (Spooky Rounding (Adapted From [DHRW16]))**.** *For moduli* $p, q \in \mathbb{N}$ *along with target value* $v \in \mathbb{Z}_q$ *and error bound, where* $p \cdot \sigma \ll q$*, consider an additive sharing* $v_1, v_2 \xleftarrow{R} \mathbb{Z}_q$ *subject to* $v_1 + v_2 = v \mod q$ *then* $\lceil v_1 \rfloor_p + \lceil v_2 \rfloor_p = \lceil v \rfloor_p$*.*

Definition 7.4 (Leftover Hash Lemma (Adapted from [ABB10]))**.** Let λ be a security parameter and n, m, q be lattice parameters where $m \geq 2n \log q$, $n \geq \lambda$ and prime $q > 2$. Then for all fixed vectors $\mathbf{e} \in \mathbb{Z}_q^m$ and $k \in \mathsf{poly}(\lambda)$, the following distributions are statistically close:

$$(\mathbf{A}, \mathbf{AR}, \mathbf{e}^\intercal \mathbf{R}) \approx_s (\mathbf{A}, \mathbf{U}, \mathbf{e}^\intercal \mathbf{R}),$$

where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}, \mathbf{R} \xleftarrow{R} \{0,1\}^{m\times k}, \mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{n\times k}$.

Assumption 7.5 (Decomposed LWE [AMR25])**.** Let λ be a security parameter and $n = n(\lambda)$, $m = m(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, and $\chi = \chi(\lambda)$ be lattice parameters. We say that the decomposed LWE assumption $\mathsf{dLWE}_{n,m,q,t,\chi,\sigma}$ assumption holds if for all PPT adversaries $\mathcal{A}$:

$$\left| \Pr\left[\mathcal{A}\left(\{\mathbf{A}_i, \mathbf{B}_j \mathbf{s}^\intercal \mathbf{A}_i \mathbf{B}_j + \mathbf{s}^\intercal \mathbf{G} \cdot \delta_{i,j} + \mathbf{e}_{i,j}^\intercal\}_{i,j\in[t]}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\{\mathbf{A}_i, \mathbf{B}_j, \mathbf{u}_{i,j}\}_{i,j\in[t]^\intercal}\right) = 1\right] \right| = \mathsf{negl}(\lambda)$$

where $\mathbf{A}_i \xleftarrow{R} \mathbb{Z}_q^{n\times k}, \mathcal{B}_j \xleftarrow{R} \mathcal{D}_\chi^{k\times m}, \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{e}_{i,j} \leftarrow \mathcal{D}_\sigma^m, \mathbf{u}_{i,j} \xleftarrow{R} \mathbb{Z}_q^m$ for $i, j \in [t]$.

Definition 7.6 (Lattice Gadget Trapdoors [Ajt96, GPV08, MP12].)**.** Let λ be a security parameter and n, m, q be lattice parameters with $m \geq 3n \log q$ and $n \geq \lambda$. There exists efficient randomized algorithms $(\mathsf{TrapGen}, \mathsf{SamplePre})$ such that:

[13] In general, we will use uniform noise in our constructions, which can clearly be sampled in NC^1

- $\mathsf{TrapGen}(1^n, 1^m, q) \to (\mathbf{A}, \mathsf{td}_{\mathbf{A}})$: On input matrix dimensions n, m and modulus q, TrapGen outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a trapdoor $\mathsf{td}_{\mathbf{A}} \in \mathbb{Z}_q^{m \times m}$.
- $\mathsf{SamplePre}(\mathbf{A}, \mathsf{td}, \mathbf{T}, \sigma) \to \mathbf{P}$: On input matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, trapdoor matrix $\mathsf{td} \in \mathbb{Z}_q^{m \times m}$, target matrix $\mathbf{T} \in \mathbb{Z}_q^{n \times \ell}$, and width parameter $\sigma > 0$, the SamplePre outputs a preimage $\mathbf{P} \in \mathbb{Z}_q^{m \times n'}$.

A gadget trapdoor satisfies the following properties:

- **Trapgen Uniformity:** In the output $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, $\mathbf{A}$ is statistically close to uniform $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$.
- **Correctness:** For all $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, all gadget trapdoors $\mathsf{td} \in \mathbb{Z}_q^{m \times m}$, and all $\mathbf{T} \in \mathbb{Z}_q^{n \times \ell}$, and all $\mathbf{P} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathsf{td}, \mathbf{T}, \sigma)$, $\mathbf{A} \cdot \mathbf{P} = \mathbf{T}$. Furthermore, in $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, $\mathsf{td}_{\mathbf{A}}$ is a gadget trapdoor $\mathbf{A} \cdot \mathsf{td} = \mathbf{G}$ with $||\mathsf{td}|| = 1$.
- **Preimage Distribution:** There exists $B = B(\lambda)$, such that for all $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, all gadget trapdoors $\mathsf{td} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{A} \cdot \mathsf{td} = \mathbf{G}$, and all $\sigma \geq m \cdot ||\mathsf{td}|| \cdot \log n$, the distributions of

  $$(\mathsf{SamplePre}(\mathbf{A}, \mathsf{td}, \mathbf{T}, \sigma), \mathbf{T}) \approx_s (\mathbf{P}, \mathbf{A} \cdot \mathbf{P})$$

  where $\mathbf{T} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell}$ and $\mathbf{P} \xleftarrow{R} \mathcal{D}_{B \cdot \sigma}^{m \times \ell}$ are statistically close.

Remark 7.7. We note the above notion of gadget trapdoors is slightly weaker than the usual definition, where the preimage distribution is explicitly required to be a specific coset of the discrete gaussian distribution with width σ. Looking ahead to Section 7.3.1, this will be in the interest of realising a simple NC^1 instantiation of preimage sampling.

Definition 7.8 (Lattice-based Homomorphic Evaluation [GSW13, BGG+14])**.** Let λ be a security parameter and n, m, q, σ be lattice parameters for $m \geq 2n \log q$. Let $C_{\ell,d}$ be a family of depth d boolean circuits $C : \{0,1\}^\ell \to \{0,1\}$. There exists efficient (deterministic) algorithms:

- $\mathsf{EvalF}(\mathbf{B}, C) \to \mathbf{B}_C$: On input matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times \ell \cdot m}$ and circuit $C \in C_{\ell,d}$, the input-independent evaluation EvalF computes matrix $\mathbf{B}_C$.
- $\mathsf{EvalFX}(\mathbf{B}, C, x) \to \mathbf{H}_{\mathbf{B},C,x}$: On input matrix $\mathbf{B}$, circuit $C \in C_{\ell,d}$, and input $x \in \{0,1\}^\ell$, the input-dependent evaluation EvalFX computes low norm matrix $\mathbf{H}_{\mathbf{B},C,x}$.

Such that the following properties hold:

- **Low Norm:** The matrix $\mathbf{H}_{\mathbf{B},C,x}$ has low norm $||\mathbf{H}_{\mathbf{B},C,x}|| \leq m^{O(d)}$.
- **Homomorphism:** For any matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times \ell \cdot m}$, circuit $C \in C_{\ell,d}$ and input $x \in \{0,1\}^\ell$, the equation

  $$(\mathbf{B} - x^\top \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{B},C,x} = \mathbf{B}_C - C(x) \cdot \mathbf{G}$$

  holds.

Definition 7.9 (Key-Homomorphic RAM Encodings [AMR25])**.** Let λ be a security parameter and n, m, q, σ be lattice parameters for $m \geq 2n \log q$. Let $\ell \in \mathsf{poly}(\lambda)$. There exists efficient (and deterministic apart from RSetup) algorithms:

- $\mathsf{RSetup}(1^\lambda) \to (\mathsf{pp}, \mathbf{Z})$: On input security parameter λ, the Setup algorithm outputs public parameter pp, and matrix $\mathbf{Z} \in \mathbb{Z}_q^{n \times t}$.
- $\mathsf{RHash}(\mathsf{pp}, x) \to \mathbf{d}$: On input public parameters pp and input $x \in \{0,1\}^\ell$, the hash algorithm deterministically computes digest vector $\mathbf{d} \in \mathbb{Z}_q^k$.
- $\mathsf{RRead}(\mathsf{pp}, \mathbf{B}, i) \to \mathbf{B}_{\langle i \rangle}^{(r)}$: On input public parameters pp, matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times k \cdot m}$, and index $i \in [\ell]$, the input-independent read function computes matrix $\mathbf{B}_{\langle i \rangle}^{(r)}$.

- $\mathsf{RXRead}(\mathsf{pp}, \mathbf{B}, i, x) \to \mathbf{H}^{(r)}_{\mathbf{B},\langle i\rangle,x}$: On input public parameters pp, matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times k\cdot m}$, index $i \in [\ell]$, and bitstring $x \in \{0,1\}^\ell$, the input-dependent read function computes matrix $\mathbf{H}^{(r)}_{\mathbf{B},\langle i\rangle,x}$.
- $\mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, S, \mathbf{B}_S) \to \mathbf{B}_S^{(w)}$: On input public parameters pp, and writing indices $S \subseteq [\ell]$, matrix $\mathbf{B}_S \in \mathbb{Z}_q^{n\times m\cdot|S|}$, the write function computes a matrix $\mathbf{B}_S^{(w)}$.
- $\mathsf{RXWrite}(\mathsf{pp}, \mathbf{B}, S, \mathbf{B}_S, x^{(w)}, x) \to \mathbf{H}^{(w)}_{\mathbf{B},S,x}$: On input public parameters pp, and writing indices $S \subseteq [\ell]$, written bits $x^{(w)} \in \{0,1\}^{|S|}$ and matrix $\mathbf{B}_S \in \mathbb{Z}_q^{n\times m\cdot|S|}$, along with input $x \in \{0,1\}^\ell$ the write function computes a matrix $\mathbf{H}^{(w)}_{\mathbf{B},S,x'}$.

Such that the following properties hold:

- **Succinctness:** The lengths of public parameters pp, matrix $\mathbf{Z}$ and digest $\mathbf{d}$ are independent of the input length $|x| = \ell$.
- **Low Norm:** The matrices $\mathbf{H}^{(r)}_{\mathbf{B},\langle i\rangle,x}, \mathbf{H}^{(w)}_{\mathbf{B},S,x}$ has low norm $||\mathbf{H}^{(r)}_{\mathbf{B},\langle i\rangle,x}, \mathbf{H}^{(w)}_{\mathbf{B},S,x}|| \le \mathsf{poly}(\lambda)$.
- **Read Homomorphism:** For any matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times k\cdot m}$, indices $i \in [\ell]$ and input $x \in \{0,1\}^\ell$, the equation
$$[\mathbf{B} - \mathsf{RHash}(x) \otimes \mathbf{G} \mid \mathbf{Z}] \cdot \mathbf{H}_{\mathbf{B},\langle i\rangle,x} = \mathbf{B}_{\langle i\rangle} - x_i \cdot \mathbf{G}$$
holds.
- **Write Homomorphism:** For any set $S = \{i_1, \ldots, i_{|S|}\} \subseteq [\ell]$, matrices $\mathbf{B} \in \mathbb{Z}_q^{n\times k\cdot m}, \mathbf{B}_S \in \mathbb{Z}_q^{n\times m\cdot|S|}$, strings $x^{(w)} \in \{0,1\}^{|S|}, x \in \{0,1\}^\ell$,
$$[\mathbf{B} - \mathsf{RHash}(x) \otimes \mathbf{G} \mid \mathbf{B}_S - x^{(w)} \otimes \mathbf{G} \mid \mathbf{Z}] \cdot \mathbf{H}^{(w)}_{\mathbf{B},S,x} = \mathbf{B}_S^{(w)} - \mathsf{RHash}(x') \otimes \mathbf{G}$$
where $x'_i = x_j^{(w)}$ for any $i_j \in S$ and $x'_i = x_i$ otherwise.
- **LHL Compatible:** For all fixed vectors $\mathbf{e} \in \mathbb{Z}_q^t$ and $k \in \mathsf{poly}(\lambda)$, the distributions
$$(\mathsf{pp}, \mathbf{Z}, \mathbf{Z}\cdot\mathbf{R}, \mathbf{e}^\top\mathbf{R}) \approx_s (\mathsf{pp}, \mathbf{Z}, \mathbf{U}, \mathbf{e}^\top\mathbf{R})$$
are statistically close, where $(\mathsf{pp}, \mathbf{Z}) \leftarrow \mathsf{RSetup}(1^\lambda), \mathbf{R} \xleftarrow{R} \{0,1\}^{t\times m}$, and $\mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{n\times t}$.
- **LWE indistinguishability:** Given
$$\{\mathsf{pp}, \mathbf{Z}, \mathbf{s}^\top\mathbf{Z} + \mathbf{e}^\top\} \approx_c \{\mathsf{pp}, \mathbf{Z}, \mathbf{t}^\top\}$$
where $(\mathsf{pp}, \mathbf{Z}) \leftarrow \mathsf{RSetup}(1^\lambda), \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{e}^\top \xleftarrow{R} \mathcal{D}_\sigma^m$ and $\mathbf{t} \xleftarrow{R} \mathbb{Z}_q^m$.

Remark 7.10. We note the above can also be instantiated with the succinct LWE assumption of [Wee24] (essentially corresponding to the matrix commitment abstraction of [Wee25]). This translates translates to a concretely smaller instantiation (where $t \approx m$ rather than $t \approx 4m^5$ using decomposed LWE), but requires a private-coin RSetup in contrast to the public-coin RSetup algorithm from decomposed LWE.

Remark 7.11. We briefly discuss the properties here that were not explicitly outlined in prior work. The LHL Compatibility follows from a straightforward inspection of the structure of the $\mathbf{Z}$ matrix. In the succinct LWE instantiation, this is a uniformly random matrix, and so corresponds exactly to the classic leftover hash lemma. In the decomposed LWE instantiation, this matrix takes the form of $\mathbf{Z} = \mathbf{A}\cdot(\mathbf{I}_{m^2} \otimes \mathbf{B}) + \mathsf{vec}(\mathbf{I}_{m^2}) \otimes \mathbf{G}$ where $\mathbf{A}$ is a random $n \times m$ matrix. This is sufficient for to apply LHL. The LWE indistinguishability property essentially corresponds exactly to the decomposed (resp. succinct) LWE assumptions.

7.2 Construction

We are now ready to present our construction of reusable privacy-free garbling.

Construction 7.12 (Succinct Reusable Privacy-Free Garbling)**.** Let λ be the security parameter and d be a bound on circuit depth. Our scheme relies on the following lattice ingredients:

- A lattice gadget trapdoor sampler $(\mathsf{TrapGen}, \mathsf{SamplePre})$ per Definition 7.6.
- Lattice-based homomorphic evaluation $(\mathsf{EvalF}, \mathsf{EvalFX})$ per Definition 7.8.
- Key-homomorphic RAM encoding $(\mathsf{RSetup}, \mathsf{RHash}, \mathsf{RRead}, \mathsf{RXRead}, \mathsf{RWrite}, \mathsf{RXWrite})$ per Definition 7.9.

We parameterize our scheme as follows:[14]

$$n \geq \lambda \quad , \quad m = 3n \log q \quad , \quad B = \lambda^{\log(\lambda)} \quad , \quad \sigma_{\mathsf{large}} = \sigma \cdot B^{d+3}$$

$$\Delta = B^{d+4} \cdot \sigma \quad , \quad p = \sigma \cdot B \quad , \quad q = \Delta \cdot p,$$

where the $\mathsf{dLWE}_{n,m,q,2m^2\lambda,\chi,\sigma}$ is hard with respect to security parameter λ, which in turn fix parameters t, k of the RAM encoding. We can now present our scheme as follows:

- $\mathsf{Setup}(1^\lambda, 1^d) \to \mathsf{pd}$: On input security parameter λ and bound on circuit depth d, the setup algorithm samples the following:
  - Sample $(\mathsf{pp}, \mathbf{Z}) \leftarrow \mathsf{RSetup}(1^\lambda)$.
  - Sample random matrix $\mathbf{B} \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$.
- $\mathsf{Enc}(x)$: For an input value $x \in \{0,1\}^\ell$, the garbler samples the following:
  - Sample input matrix $\mathbf{B}_{\mathsf{inp}} \xleftarrow{R} \mathbb{Z}_q^{n \times m \cdot \ell}$.
  - Sample secret $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$ and trapdoored matrix $(\mathbf{A}, \mathsf{td}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
  - To compute error vectors, sample and set

$$\mathbf{e}_0 \xleftarrow{R} \mathcal{D}_\sigma^t \, , \, \mathbf{R}_\mathbf{A} \xleftarrow{R} \{0,1\}^{t \times m} \, , \, \mathbf{R}_\mathbf{B} \xleftarrow{R} \{0,1\}^{t \times m \cdot k} \, , \, \mathbf{R}_{\mathsf{inp}} \xleftarrow{R} \{0,1\}^{t \times m \cdot \ell}$$

$$\mathbf{e}_1^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_\mathbf{A} \, , \, \mathbf{e}_2^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_\mathbf{B} \, , \, \mathbf{e}_3^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_{\mathsf{inp}}.$$

  - Set garbled input as:

$$\mathbf{C}_\mathbf{Z} = \mathbf{s}^\intercal \cdot \mathbf{Z} + \mathbf{e}_0^\intercal \quad , \quad \mathbf{C}_\mathbf{A} = \mathbf{s}^\intercal \cdot \mathbf{A} + \mathbf{e}_1^\intercal$$

$$\mathbf{C}_\mathbf{B} = \mathbf{s}^\intercal \cdot (\mathbf{B} - \mathsf{RHash}(0) \otimes \mathbf{G}) + \mathbf{e}_2^\intercal \text{[15]} \quad , \quad \mathbf{C}_x = \mathbf{s}^\intercal \cdot (\mathbf{B}_{\mathsf{inp}} - x^\intercal \otimes \mathbf{G}) + \mathbf{e}_3^\intercal.$$

  - We treat matrix $\mathbf{B}_{\mathsf{inp}}$ as $[\![x]\!]_G$ and $(x, \mathbf{B}_{\mathsf{inp}}, \mathbf{C}_x)$ as the $[\![x]\!]_E$, which is sent to the evaluator. The other components $(\mathbf{C}_\mathbf{Z}, \mathbf{C}_\mathbf{A}, \mathbf{C}_\mathbf{B})$ are will be used as auxiliary information as part of the garbled input.
- $\mathsf{HGate}(\mathsf{op}, [\![x]\!], [\![y]\!])$: The gate evaluation correspond exactly to lattice-based homomorphic evaluation of Definition 7.8.
  - $\mathsf{HGate}_G(\mathsf{op}, \mathbf{B}_x, \mathbf{B}_y) \to \mathbf{B}_{\mathsf{op}(x,y)}$: The garbler's gate-evaluation algorithm corresponds to the input-independent homomorphic evaluation $[\![\mathsf{op}(x,y)]\!]_G = \mathbf{B}_{\mathsf{op}(x,y)} \leftarrow \mathsf{EvalF}([\mathbf{B}_x \mid \mathbf{B}_y], \mathsf{op})$.
  - $\mathsf{HGate}_E(\mathsf{op}, (x, \mathbf{B}_x, \mathbf{C}_x), (y, \mathbf{B}_y, \mathbf{C}_y)) \to \mathbf{C}_{\mathsf{op}(x,y)}$:The evaluators gate-evaluation algorithm corresponds to the input-dependent homomorphic evaluation
    - Compute input-independent evaluation $\mathbf{B}_{\mathsf{op}(x,y)} \leftarrow \mathsf{EvalF}([\mathbf{B}_x \mid \mathbf{B}_y], \mathsf{op})$.

[14] We choose these parameters for simplicity and are not optimized for poly/polylog factors in λ.

[15] In the [AMR25, Wee25] instantiations of RAM encodings, $\mathsf{RHash}(0) = 0$, so this is simply $\mathbf{s}^\intercal \cdot \mathbf{B} + \mathbf{e}_2^\intercal$

* Compute input-dependent homomorphic evaluation $\mathbf{H}_{\mathsf{op}(x,y)} = \mathsf{EvalFX}([\mathbf{B}_x \mid \mathbf{B}_y], \mathsf{op}, (x \mid\mid y))$.
* Set encoding as $[\![\mathsf{op}(x,y)]\!]_E = (\mathsf{op}(x,y), \mathbf{B}_{\mathsf{op}(x,y)}, [\mathbf{C}_x \mid \mathbf{C}_y] \cdot \mathbf{H}_{\mathsf{op}(x,y)})$.

- $\mathsf{Hash}(\hat{C}, x) \to h_x$: This corresponds exactly to RHash function of key-homomorphic RAM encodings of Definition 7.9. In the reusable setting, the garbler will not need his share of $[\![x]\!]_G$ to compute $\mathsf{HRead}_G, \mathsf{HCompress}_G$, and so we omit these as inputs below.
  - $\mathsf{HRead}_G(\mathbf{B}_{h_x}, i)$: The garbler's read algorithm corresponds to input-independent read $[\![x_i]\!]_G = \mathbf{B}_{x_i} \leftarrow \mathsf{RRead}(\mathsf{pp}, \mathbf{B}_{h_x}, i)$.
  - $\mathsf{HRead}_E((h_x, \mathbf{B}_{h_x}, \mathbf{C}_{h_x}), i, (x, \mathbf{B}_x, \mathbf{C}_x))$: The evaluator's read algorithm corresponds to input-dependent read
    * Compute input-independent read $\mathbf{B}_{x_i} \leftarrow \mathsf{RRead}(\mathsf{pp}, \mathbf{B}_{h_x}, i)$.
    * Compute input-dependent $\mathbf{H}_{x_i} = \mathsf{RXRead}(\mathsf{pp}, \mathbf{B}_{h_x}, i, x)$.
    * Set encoding $[\![x_i]\!]_E = (x_i, \mathbf{B}_{x_i}, [\mathbf{C}_{h_x} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{H}_{x_i})$.
  - $\mathsf{HCompress}_G(\mathbf{B}_x)$: The garbler's compress algorithm corresponds to input-independent write $[\![h_x]\!]_G = \mathbf{B}_{h_x} \leftarrow \mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, [|x|], \mathbf{B}_x)$.
  - $\mathsf{HCompress}_E(x, \mathbf{B}_x, \mathbf{C}_x)$: The evaluator's read algorithm corresponds to input-dependent write
    * Compute input-independent write
    $$\mathbf{B}_{h_x} \leftarrow \mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, [|x|], \mathbf{B}_x).$$
    * Compute input-dependent $\mathbf{H}_{h_x} = \mathsf{RXWrite}(\mathsf{pp}, \mathbf{B}, [|x|], \mathbf{B}_x, 0, x)$.
    * Set encoding $[\![h_x]\!]_E = (h_x, \mathbf{B}_{x_i}, [\mathbf{C}_{\mathbf{B}} \mid \mathbf{C}_x \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{H}_{h_x})$.

- $\mathsf{Rekey}([\![x]\!]) \to [\![x]\!]'$:
  - $\mathsf{RekeySamp}(\mathsf{gk}) \to [\![x]\!]'_G$: Sample fresh random matrix $\mathbf{B}' \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$, which will be set as $[\![x]\!]'_G$.
  - $\mathsf{RekeyGen}([\![x]\!]_G, [\![x]\!]'_G) \to \mathsf{rk}$: Parsing $[\![x]\!]_G = \mathbf{B}_{h_x}$ and $[\![x]\!]'_G = \mathbf{B}'$, sample
  $$\mathsf{rk} \leftarrow \mathsf{SamplePre}([\mathbf{A} \mid \mathbf{Z}], \begin{bmatrix} \mathsf{td}_{\mathbf{A}} \\ 0 \end{bmatrix}, \mathbf{B}' - \mathbf{B}_{h_x}, \sigma),$$
  which is sent to the evaluator.
  - $\mathsf{RekeyEval}([\![x]\!]_E, \mathsf{rk}) \to [\![x]\!]'_E$: Evaluator parses their share $[\![x]\!]_E = (h_x, \mathbf{B}_{h_x}, \mathbf{C}_{h_x})$ computes their new share as $(h_x, \mathbf{B}', \mathbf{C}_{h_x} + [\mathbf{C}_{\mathbf{A}} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathsf{rk})$[16].

- **Output Phase:** The garbler samples $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{e}_{\mathsf{large}} \leftarrow \mathcal{D}_{\sigma_{\mathsf{large}}}$ and rounding shifts $\delta \xleftarrow{R} \mathbb{Z}_q$.
  - The garbler parses their share of the output wire $[\![y_{\mathsf{out}}]\!]_G = \mathbf{B}_{\mathsf{out}}$. They output $\mathbf{u}, \delta$, decoding key as $\mathsf{dk} = \lfloor (\mathbf{s}^\top \cdot \mathbf{B}_{\mathsf{out}} + \mathbf{e}_{\mathsf{large}}) \cdot \mathbf{G}^{-1}(\mathbf{u}) \rceil + \delta \rceil_p$ and authentication key $\mathsf{auk} = \lfloor \mathbf{s}^\top [\mathbf{B}_{\mathsf{out}} \mid \mathbf{B}_{\mathsf{out}} \cdot \mathbf{G}^{-1}(\mathbf{u})] + \delta \cdot \mathbf{1}_{m+1}^\top \rceil_p, \lfloor \mathbf{s}^\top [\mathbf{G} \mid \mathbf{u}] \rceil_p$.
  - The evaluator parses their share of the output wire $[\![y_{\mathsf{out}}]\!]_E = (y_{\mathsf{out}}, \mathbf{B}_{\mathsf{out}}, \mathbf{C}_{\mathsf{out}})$. Compute output label as $\left\lfloor [\mathbf{C}_{\mathsf{out}} \mid \mathbf{C}_{\mathsf{out}} \cdot \mathbf{G}^{-1}(\mathbf{u})] + \delta \cdot \mathbf{1}_{m+1} \right\rceil_p$

Remark 7.13 (Succinct Homomorphic Encodings)**.** For notational convenience, when arguing correctness and security of our scheme, we will use $(\mathsf{SEvalF}, \mathsf{SEvalFX})$ to denote succinct homomorphic evaluation performed directly on the RAM encoding(s) of Definition 7.9. This corresponds directly to composing read, homomorphic evaluation of Definition 7.8, and write in both the input-dependent and input-independent settings, and the HRead, HGate, HCompress operations when used in the above garbling construction Construction 7.12.

[16] Note that rk allows the evaluator to efficiently compute $\mathbf{B}'$ with $\mathbf{A}, \mathbf{Z}, \mathbf{B}_{h_x}$, which are all known.

- $\mathsf{SEvalF}(\mathsf{pp}, \mathbf{B}, f) \to \mathbf{B}_f$: On input matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times k \cdot m \cdot t}$ where $\mathbf{B} = [\mathbf{B}_1 \mid \ldots \mid \mathbf{B}_t]$ and function $f : \{0,1\}^{\ell_1} \times \ldots \times \{0,1\}^{\ell_t} \to \{0,1\}^{\ell'} \in \mathcal{F}_d$, the input-independent evaluation EvalF computes matrix $\mathbf{B}_f \in \mathbb{Z}_q^{n \times k \cdot m}$.
- $\mathsf{SEvalFX}(\mathsf{pp}, \mathbf{B}, f, x) \to \mathbf{H}_{\mathbf{B},f,x}$: On input matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times k \cdot m \cdot t}$ where $\mathbf{B} = [\mathbf{B}_1 \mid \ldots \mid \mathbf{B}_t]$, function $f \in \{0,1\}^{\ell_1} \times \cdots \times \{0,1\}^{\ell_t} \to \{0,1\}^{\ell'} \in \mathcal{F}_d$, and inputs $x = (x_1, \ldots x_t)$ for $x_i \in \{0,1\}^{\ell_i}$ for $i \in [t]$, the input-dependent evaluation EvalFX computes low norm matrix $\mathbf{H}_{\mathbf{B},f,x}$.

We note that that this composition inherits the low norm properties of $\mathbf{H}_f, \mathbf{K}_f$ by simply multiplying together the $\mathbf{H}$ matrices as well as respects the homomorphism:

$$[\mathbf{B}_1 - \mathsf{RHash}(x_1) \otimes \mathbf{G} \mid \ldots \mid \mathbf{B}_t - \mathsf{RHash}(x_t) \mid \mathbf{Z}] \cdot \mathbf{H}_{\mathbf{B},f,x} = \mathbf{B}_f - \mathsf{RHash}(f(x_1, \ldots x_t)) \otimes \mathbf{G}.$$

7.3 Garbler Depth Complexity

We follow Section 4.1 in instantiating a low-depth version of the garbler's computation. We note it is easy to see that bit decomposition, as well as elementary (modular) arithmetic [CDL01] and matrix operations (i.e. addition, multiplication, tensor product, bit decomposition) over $\mathbb{Z}_q$ for q of poly-bounded length can all be implemented in $\mathsf{NC1}$. This is sufficient to implement the majority of our garbling scheme, including [BGG+14] lattice homomorphisms. We discuss the remaining ingredients below.

7.3.1 Log-Depth Preimage Sampling

We show a simple gadget trapdoor which is easy to see is in NC^1 and suffices for our applications. We can take constant B to be any function $\lambda^{\omega(1)}$.

- $\mathsf{TrapGen}(1^n, 1^m, q) \to (\mathbf{A}, \mathsf{td}_\mathbf{A})$: On input matrix dimensions n, m with $m \geq 3n \log q$ and modulus q,
  - Sample matrix $\mathbf{B} \xleftarrow{R} \mathbb{Z}_q^{n \times 2n \log q}$
  - Sample matrix $\mathbf{R} \xleftarrow{R} \{0,1\}^{(2n \log q) \times (m - 2n \log q)}$
  - Output $\mathbf{A} = [\mathbf{B} \mid \mathbf{B} \cdot \mathbf{R} - \mathbf{G}]$ and $\mathsf{td}_\mathbf{A} = \begin{bmatrix} \mathbf{R} \\ -\mathbf{I} \end{bmatrix}$.
- $\mathsf{SamplePre}(\mathbf{A}, \mathsf{td}, \mathbf{T}, \sigma)$: On input matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, trapdoor $\mathsf{td} \in \mathbb{Z}_q^{m \times n \log q}$, target matrix $\mathbf{T} \in \mathbb{Z}_q^{n \times k}$ and parameter σ,
  - Sample $\mathbf{R} \xleftarrow{R} [\sigma \cdot B]^{m \times k}$ and set $\mathbf{V} = \mathbf{A} \cdot \mathbf{R}$.
  - Output $\mathsf{td} \cdot \mathbf{G}^{-1}(\mathbf{T} - \mathbf{V}) + \mathbf{R}$.

  To see this construction meets the definition of gadget trapdoor Definition 7.6, we can first observe that correctness follows from the relation

$$\begin{aligned} \mathbf{A} \cdot \mathbf{P} &= \mathbf{A} \cdot (\mathsf{td} \cdot \mathbf{G}^{-1}(\mathbf{T} - \mathbf{V}) + \mathbf{R}) \\ &= (\mathbf{A} \cdot \mathsf{td}) \cdot \mathbf{G}^{-1}(\mathbf{T} - \mathbf{V}) + \mathbf{A} \cdot \mathbf{R} \\ &= \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{T} - \mathbf{V}) + \mathbf{V} \\ &= \mathbf{T} - \mathbf{V} + \mathbf{V} \\ &= \mathbf{T} \end{aligned}$$

  Uniformity follows from the leftover hash lemma, while the preimage norm is apparent by construction. Finally, to observe preimage indistinguishability, we will show the distribution of $(\mathbf{A}, \mathbf{P}, \mathbf{T})$ generated through preimage sampling is statistically close to sampling $\mathbf{P}$ as a uniform matrix $\in [\sigma \cdot B]^{m \times k}$ and setting $\mathbf{T} = \mathbf{A} \cdot \mathbf{P}$. Since the latter distribution is independent of any trapdoor to $\mathbf{A}$, preimage indistinguishability follows.

  1. This is the real preimage sampling experiment, where $\mathbf{T} \xleftarrow{R} \mathbb{Z}_q^{n \times k}$ and $\mathbf{P} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathsf{td}, \mathbf{T}, \sigma)$.

2. In this experiment, sample $\mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{n\times k}$ and computing $\mathbf{P} = \mathsf{td} \cdot \mathbf{G}^{-1}(\mathbf{U}) + \mathbf{R}$ and $\mathbf{T} = \mathbf{A} \cdot \mathbf{P}$.
3. In this experiment, sample $\mathbf{P} \xleftarrow{R} [\sigma \cdot B]^{m\times k}$.

We can see experiments (1) and (2) above are equivalent, by implicitly setting $\mathbf{U} = (\mathbf{T} - \mathbf{V})$. and steps (2) and (3) are statistically close by a simple noise-smudging argument as $\sigma \cdot B \gg \sigma$.

7.3.2 Log-Depth Key-Homomorphic RAM Encoding

Recall at a high level [AMR25] RAM encodings RHash is encoded as recursively as linear 'Merkle hash' of the input, where each level can be implemented through log-depth elementary matrix operations. Thus to ensure the garbler's overall computation is log-depth, we are restricted to performing an only *constant* number of recursions. This can be accomplished by setting the 'arity' of Hash to λ (parameter β in [AMR25]). This incurs a λ factor in the size of the hash public parameters compared to the usual choice of choosing this arity to be a constant > 1. However, now for any circuit family of polynomially bounded size, because there exists constant $c_{\max}$ such that $|C| \le \lambda^{c_{\max}}$, we can hash all the wires of C in a string of length $\lambda^{c_{\max}}$, which can be done in depth $c_{\max} \cdot \log(\lambda, d)$.

7.4 Succinctness and Decomposability

Recall the garbling material consists of a set of public parameters pp of RSetup and a series of matrices $\{\mathsf{rk}^{(i)}\}_{i\in[d]}$, $\mathbf{C_A}, \mathbf{C_Z}, \mathbf{C}_x, \mathsf{dk}$ over $\mathbb{Z}_q$ dimensions dependent on n, m, t, k. From inspection of our parameter instantiation we can see $q \in 2^{\mathsf{poly}(\lambda,d)}$, and thus $n, m \in \mathsf{poly}(\lambda, d)$. Recall that t, k are also of size $\mathsf{poly}(\lambda, d)$ by the succinctness property of the key-homomorphic hash.

Note that the garbled input labels consists of $(\mathbf{C_A}, \mathbf{C_Z}, \mathbf{C_B}, \mathbf{C}_x, \mathbf{B}_{\mathsf{inp}})$, where $\mathbf{C_A}, \mathbf{C_Z}, \mathbf{C_B}, \mathbf{B}_{\mathsf{inp}}$ are computable independent of x, and $\mathbf{C}_x$ is computed as

$$\begin{aligned}
\mathbf{C}_x &= \mathbf{s}^\top \cdot (\mathbf{B}_{\mathsf{inp}} - x^\top \otimes \mathbf{G}) + \mathbf{e}_3^\top \\
&= \mathbf{s}^\top \cdot ([\mathbf{B}^{(r)}_{\langle 1\rangle} \mid \cdots \mid \mathbf{B}^{(r)}_{\langle \ell\rangle}] - x^\top \otimes \mathbf{G}) + \mathbf{e}_3^\top \\
&= \left[\mathbf{s}^\top \cdot \left(\mathbf{B}^{(r)}_{\langle 1\rangle} - x_1 \cdot \mathbf{G}\right) + \mathbf{e}^\top_{3,1} \mid \cdots \mid \mathbf{s}^\top \cdot \left(\mathbf{B}^{(r)}_{\langle \ell\rangle} - x_\ell \cdot \mathbf{G}\right) + \mathbf{e}^\top_{3,\ell}\right]
\end{aligned}$$

where each $\mathbf{e}_{2,i}$ are simply length m the blocks of the error vector $\mathbf{e}_3$. It is easy to see each 'block' of $\mathbf{C}_x$ depends on exactly one bit x_i of x, giving us decomposability.

7.5 Correctness

To show correctness of our scheme, it suffices to show it conforms to our notion of algebraic labeling - i.e. taking the garbler's authentication key as $(\mathbf{r}^{(\mathrm{out})}, \mathbf{s}^{(\mathrm{out})})$, the decoding key is the last entry of $\mathbf{r}^{(\mathrm{out})}$ and the evaluator's label takes the form of $(\mathbf{r}^{((\mathrm{out})} + C(x) \cdot \mathbf{s}^{(\mathrm{out})})$.

Inspecting our scheme, recall from our authentication key that

$$\mathbf{r}^{(\mathrm{out})} = \lfloor \mathbf{s}^\top [\mathbf{B}_{\mathsf{out}} \mid \mathbf{B}_{\mathsf{out}} \cdot \mathbf{G}^{-1}(\mathbf{u})] + \delta \cdot \mathbf{1}^\top_{m+1} \rceil_p \quad , \quad \mathbf{s}^{(\mathrm{out})} = \left\lfloor \mathbf{s}^\top [\mathbf{G} \mid \mathbf{u}] \right\rceil_p$$

With this in mind, let $(h^{(i)}, \mathbf{B}^{(i)}, \mathbf{C}^{(i)})$ be the evaluator's share of the hash of all depth i wires (Where $\mathbf{B}^{(0)}$ consists of the hash of the input encoding). Similarly, let $x^{(i)}$ be the value of all depth i wires (with $x = x^{(0)}$). We will inductively bound the distance of

$$||\mathbf{s}^\top \cdot (\mathbf{B}^{(i)} - h^{(i)} \otimes \mathbf{G}) - \mathbf{C}^{(i)}|| \le B^{i+1} \cdot \sigma$$

Let $\mathbf{e}^{(i)}$ refer to this difference. To begin with the base case, recall $(h^{(0)}, \mathbf{B}^{(0)}, \mathbf{C}^{(0)})$ is computed from the garbled input $(x, \mathbf{B}_{\mathsf{inp}}, \mathbf{C}_x)$ as:

$$h^{(0)} = \mathsf{RHash}(x) \quad , \quad \mathbf{B}^{(0)} \leftarrow \mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, [\ell], \mathbf{B}_{\mathsf{inp}}) \quad , \quad \mathbf{C}^{(0)} = [\mathbf{C_B} \mid \mathbf{C}_x \mid \mathbf{C_Z}] \cdot \mathbf{H}^{(0)}$$

where $\mathbf{H}^{(0)} \leftarrow \mathsf{RXWrite}(\mathsf{pp}, \mathbf{B}, [\ell], \mathbf{B}_{\mathsf{inp}}, x, 0)$. From the write homomorphism of $\mathbf{H}$, this equation for $\mathbf{C}^{(0)}$ can be rewritten as

$$\begin{aligned}[\mathbf{C}_{\mathbf{B}} \mid \mathbf{C}_x \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{H}^{(0)} &= \mathbf{s}^\intercal [\mathbf{B} \mid \mathbf{B}_{\mathsf{inp}} \mid \mathbf{Z}] \cdot \mathbf{H}^{(0)} + [\mathbf{e}_2{}^\intercal \mid \mathbf{e}_3^\intercal \mid \mathbf{e}_0^\intercal] \cdot \mathbf{H}^{(0)} \\ &= \mathbf{s}^\intercal (\mathbf{B}^{(0)} - h^{(0)} \otimes \mathbf{G}) + [\mathbf{e}_2{}^\intercal \mid \mathbf{e}_3^\intercal \mid \mathbf{e}_0^\intercal] \cdot \mathbf{H}^{(0)}\end{aligned}$$

Recall from our construction we can bound the entries of $\mathbf{e}_0, \mathbf{e}_2, \mathbf{e}_3$ with $\sigma \cdot t \in \sigma \cdot \mathsf{poly}(\lambda)$ and we can bound the entries of $\mathbf{H}^{(0)}$ with $\mathsf{poly}(\lambda)$ by the low norm property of Definition 7.9, so the combined error vector has entries bound by $\sigma \cdot \mathsf{poly}(\lambda) \ll \sigma \cdot B$.

Next, recall $\mathbf{C}^{(i)}$ is derived from $\mathbf{C}^{(0)}, \ldots, \mathbf{C}^{(i-1)}$, the evaluator first set

$$\mathbf{C}_{C_i}^{(i-1)} = [\mathbf{C}^{(0)} \mid \ldots \mid \mathbf{C}^{(i-1)} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{H}_{C_i} \tag{7.1}$$

then computes

$$\mathbf{C}^{(i)} = \mathbf{C}_{C_i}^{(i-1)} + [\mathbf{C}_{\mathbf{A}} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathsf{rk}^{(i)} \tag{7.2}$$

where $\mathbf{H}_{C_i} \leftarrow \mathsf{SEvalFX}(\mathsf{pp}, [\mathbf{B}^{(0)} \mid \ldots \mid \mathbf{B}^{(i-1)}], C_i, C_{[i-1]}(x))$ and $\mathsf{rk}^{(i)}$ is sampled as a preimage of $\mathbf{B}^{(i)} - \mathbf{B}_{C_i}^{(i-1)}$ relative to $[\mathbf{A} \mid \mathbf{Z}]$ with width σ. Substituting $\mathbf{C}_{\mathbf{A}} = \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}_0^\intercal$, $\mathbf{C}_{\mathbf{Z}} = \mathbf{s}^\intercal \mathbf{Z} + \mathbf{e}_1^\intercal$ and $\mathbf{C}^{(i')} = \mathbf{s}^\intercal \mathbf{B}^{(i)} + \mathbf{e}_{(i')}^\intercal$ for $i' \in [0, i-1]$, we can rewrite Eq. (7.1)

$$\begin{aligned}\mathbf{C}_{C_i}^{(i-1)} = [\mathbf{C}^{(0)} \mid \ldots \mid \mathbf{C}^{(i-1)} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{H}_{C_i} &= \mathbf{s}^\intercal \cdot [\mathbf{B}^{(0)} \mid \ldots \mid \mathbf{B}^{(i-1)} \mid \mathbf{Z}] \cdot \mathbf{H}_{C_i} + [\mathbf{e}_{(0)}^\intercal \mid \ldots \mid \mathbf{e}_{(i-1)}^\intercal \mid \mathbf{e}_1^\intercal] \cdot \mathbf{H}_{C_i} \\ &= \mathbf{s}^\intercal \cdot \mathbf{B}_{C_i}^{(i-1)} + \mathbf{e}_{C_i}^{(i-1)^\intercal}\end{aligned}$$

relying on lattice homomorphism from Remark 7.13. Since $||\mathbf{H}_{C_i}|| \le m^{O(d)}$ where $m \in \mathsf{poly}(\lambda)$ and C_i is depth 1, we can bound the magnitude of the new error vector $||\mathbf{e}_{C_i}^{(i-1)^\intercal}|| \le B^i \cdot \sigma \cdot \mathsf{poly}(\lambda)$. Now we can substitute Eq. (7.2) as

$$\begin{aligned}\mathbf{C}_{C_i}^{(i-1)} + [\mathbf{C}_{\mathbf{A}} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathsf{rk}^{(i)} &= \mathbf{s}^\intercal \mathbf{B}_{C_i}^{(i-1)} + \mathbf{e}_{C_i}^{(i-1)^\intercal} + \mathbf{s}^\intercal [\mathbf{A} \mid \mathbf{Z}] \cdot \mathsf{rk}^{(i)} + [\mathbf{e}_0^\intercal \mid \mathbf{e}_1^\intercal] \cdot \mathsf{rk}^{(i)} \\ &= \mathbf{s}^\intercal (\mathbf{B}_{C_i}^{(i-1)} + (\mathbf{B}^{(i)} - \mathbf{B}_{C_i}^{(i-1)})) + \mathbf{e}_{C_i}^{(i-1)^\intercal} + [\mathbf{e}_0^\intercal \mid \mathbf{e}_1^\intercal] \cdot \mathsf{rk}^{(i)} \\ &= \mathbf{s}^\intercal \cdot \mathbf{B}^{(i)} + \mathbf{e}^{(i)}\end{aligned}$$

Recall that from construction $||\mathsf{rk}^{(i)}|| \le B \cdot \sigma$ and $||\mathbf{e}_0, \mathbf{e}_1|| \le \sigma$ so we can bound the new error vector $||\mathbf{e}^{(i)}|| \le B \cdot \sigma \cdot \mathsf{poly}(\lambda) + B^i \cdot \sigma \cdot \mathsf{poly}(\lambda)$. Since B is superpolynomial, we can loosely upper bound this quantity by $B^{i+1} \cdot \sigma$.

This allows us to conclude $\mathbf{C}^{(d)} = \mathbf{s}^\intercal \cdot \mathbf{B}^{(d)} + \mathbf{e}^{(d)}$ for $||\mathbf{e}^{(d)}|| \le B^{d+1} \cdot \sigma$. Now all that remains to compute $\mathbf{C}_{\mathsf{out}}$ is a $\mathsf{HRead}(\cdot)$ on the value of the output wire. Using an analogous argument, we can bound $||\mathbf{C}_{\mathsf{out}} - \mathbf{s}^\intercal(\mathbf{B}_{\mathsf{out}} - C(x) \cdot \mathbf{G})|| \le B^{d+2} \cdot \sigma$. Now to argue correctness, we can simply observe that we chose our rounding modulus p such that $(q/p) = \Delta \gg B^{d+2} \cdot \sigma$, which means that $\lfloor \mathbf{C}_{\mathsf{out}} + \delta \cdot \mathbf{1}_m \rceil_p = \lfloor \mathbf{s}^\intercal \cdot \mathbf{B} + \delta \cdot \mathbf{1}_m^\intercal \rceil_p - \lfloor \mathbf{s}^\intercal \cdot C(x) \cdot \mathbf{G} \rceil_p$ with $1 - \mathsf{negl}(\lambda)$ probability by Lemma 7.3 (with th rounding shift). Similarly, we can reason that since $\Delta \gg ||\mathbf{e}_{\mathsf{large}}||$ our decoding key $(\mathbf{s}^\intercal \cdot \mathbf{B}_{\mathsf{out}} + \mathbf{e}_{\mathsf{large}}^\intercal) \cdot \mathbf{G}^{-1}(\mathbf{u})$ will round to $\lfloor \mathbf{s}^\intercal \cdot \mathbf{B}_{\mathsf{out}} \cdot \mathbf{G}^{-1}(\mathbf{u}) \rceil_p$ with overwhelming probability.

7.6 Security

Below, we will prove the one-wayness of Construction 7.12, which by Lemma 3.3 suffices to show authenticity.

Theorem 7.14 (One-Wayness). *Assuming decomposed LWE, garbling scheme Construction 7.12 satisfies selective reusable one-wayness as an algebraic garbling scheme.*

Proof. Consider the following sequence of hybrids:

- $\mathsf{Hyb}_{\mathsf{real}}$: This is the real selective one-wayness experiment instantiated with Construction 7.12.
    1. The adversary $\mathcal{A}$ specifies circuit C and input x^* to the challenger.

2. The challenger samples the following:
   (a) Setup material $(\mathsf{pp}, \mathbf{Z}) \leftarrow \mathsf{RSetup}(1^\lambda)$
   (b) Randomizing binary matrices:

$$\mathbf{R}_\mathbf{A} \xleftarrow{R} \{0,1\}^{t \times m} \,,\, \mathbf{R}_\mathbf{B} \xleftarrow{R} \{0,1\}^{t \times m \cdot k} \,,\, \mathbf{R}_{\mathsf{inp}} \xleftarrow{R} \{0,1\}^{t \times m \cdot \ell}$$

   (c) Trapdoor matrix $(\mathbf{A}, \mathsf{td}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
   (d) Encoding matrices $\mathbf{B} \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$, $\mathbf{B}_{\mathsf{inp}} \xleftarrow{R} \mathbb{Z}_q^{n \times m \cdot \ell}$ and decoding vector $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$.
   (e) LWE secret and error vectors :

$$\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n \quad , \quad \mathbf{e}_0 \xleftarrow{R} \mathcal{D}_\sigma^t, \quad , \quad \mathbf{e}_{\mathsf{large}} \xleftarrow{R} \mathcal{D}_{\sigma_{\mathsf{large}}}^{m \cdot k}$$

$$\mathbf{e}_1^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_\mathbf{A} \quad , \quad \mathbf{e}_2^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_\mathbf{B} \quad , \quad \mathbf{e}_3^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_{\mathsf{inp}}.$$

   (f) Rekeying material for each $i \in [d]$:

$$\mathbf{B}^{(0)} \leftarrow \mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, [\ell], \mathbf{B}_{\mathsf{inp}}) \quad , \quad \mathbf{B}^{(i)} \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$$

$$\mathbf{B}_{C_i}^{(i-1)} \leftarrow \mathsf{SEvalF}(\mathsf{pp}, [\mathbf{B}^{(0)} \mid \ldots \mid \mathbf{B}^{(i-1)}], C_i) \quad , \quad \mathsf{rk}^{(i)} \leftarrow \mathsf{SamplePre}\left([\mathbf{A} \mid \mathbf{Z}], \begin{bmatrix} \mathsf{td}_\mathbf{A} \\ 0 \end{bmatrix}, \mathbf{B}^{(i)} - \mathbf{B}_{C_i}^{(i-1)}, 1\right)$$

   (g) Rounding shift $\delta \xleftarrow{R} \mathbb{Z}_q$.
3. The challenger computes the adversary's view as follows:
   (a) Set public data $\mathsf{pd} = (\mathsf{pp}, \mathbf{Z}, \mathbf{B}, \delta)$.
   (b) Set garbled circuit and public garbling key $\hat{C} = \mathsf{gk} = \{\mathsf{rk}^{(i)}\}_{i \in [d]}$.
   (c) Set garbled input label $L_x = (\mathbf{C}_\mathbf{A}, \mathbf{C}_\mathbf{Z}, \mathbf{C}_\mathbf{B}, \mathbf{C}_x, \mathbf{B}_{\mathsf{inp}})$ where

$$\mathbf{C}_\mathbf{Z} = \mathbf{s}^\intercal \cdot \mathbf{Z} + \mathbf{e}_0^\intercal \quad , \quad \mathbf{C}_\mathbf{A} = \mathbf{s}^\intercal \cdot \mathbf{A} + \mathbf{e}_1^\intercal$$

$$\mathbf{C}_\mathbf{B} = \mathbf{s}^\intercal \cdot (\mathbf{B} - \mathsf{RHash}(0) \otimes \mathbf{G}) + \mathbf{e}_2^\intercal \quad , \quad \mathbf{C}_x = \mathbf{s}^\intercal \cdot (\mathbf{B}_{\mathsf{inp}} - x^\intercal \otimes \mathbf{G}) + \mathbf{e}_3^\intercal.$$

   (d) Set decoding key $\mathsf{dk} = \left\lfloor \left(\mathbf{s}^\intercal \cdot \mathbf{B}_{\mathsf{out}} + \mathbf{e}_{\mathsf{large}}^\intercal\right) \cdot \mathbf{G}^{-1}(\mathbf{u}) + \delta \right\rceil_p$ where

$$\mathbf{B}_{\mathsf{out}} \leftarrow \mathsf{RRead}(\mathsf{pp}, \mathbf{B}^{(d)}, |C|^{17})$$

4. The adversary outputs $\mathbf{s}'$ and wins if $\mathbf{s}' = \lfloor \mathbf{s}^\intercal [\mathbf{G} \mid \mathbf{u}] \rceil_p$.

- Hyb_0: In this experiment, the adversary wins if it outputs any $\mathbf{s}'$ which satisfies the following:

  4. The adversary outputs $\mathbf{s}' \in \mathbb{Z}_p^{m+1}$. Recover candidate of $\mathbf{s}$ as $\mathbf{s}_q \in \mathbb{Z}_q^n$ where we set the ith bit of each entry of $\mathbf{s}_q$ as $\lceil \mathbf{s}'^\intercal \cdot \mathbf{G}^{-1}(\mathbf{I} \cdot p \cdot 2^{i-1}) \rfloor_2$. The adversary wins if $||\mathbf{s}_q^\intercal \cdot \mathbf{Z} - \mathbf{C}_\mathbf{Z}|| \leq \sigma$.

- $\mathsf{Hyb}_{1,i}$: In this experiment, the challenger samples the rekeying keys without using $\mathsf{td}_\mathbf{A}$:

  2.(f) Rekeying material for each $i \in [d]$:

$$\mathbf{B}^{(0)} \leftarrow \mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, [\ell], \mathbf{B}_{\mathsf{inp}}) \quad , \quad \mathsf{rk}^{(i)} \xleftarrow{R} \mathcal{D}_B^{(m+t) \times k \cdot m}$$

$$\mathbf{B}_{C_i}^{(i-1)} \leftarrow \mathsf{SEvalF}(\mathsf{pp}, [\mathbf{B}^{(0)} \mid \ldots \mid \mathbf{B}^{(i-1)}], C_i) \quad , \quad \mathbf{B}^{(i)} = [\mathbf{A} \mid \mathbf{Z}] \cdot \mathsf{rk}^{(i)} + \mathbf{B}_{C_i}^{(i-1)}$$

- Hyb_2: In this experiment, the challenger samples $\mathbf{A}$ as a random matrix.

  2.(c) ~~Trapdoor~~ matrix $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$.

[17] We will use the 'last' wire of C as the output wire

- Hyb_3: In this experiment, the challenger derives matrices $\mathbf{A}, \mathbf{B}, \mathbf{B}_{\mathsf{inp}}$ and vector $\mathbf{u}$ from $\mathbf{Z}$.

  2.(c) Matrix $\mathbf{A} = \mathbf{Z} \cdot \mathbf{R}_{\mathbf{A}}$.

  2.(d) Encoding matrices $\mathbf{B} = \mathbf{Z} \cdot \mathbf{R}_{\mathbf{B}} + \mathsf{RHash}(0) \otimes \mathbf{G}, \mathbf{B}_{\mathsf{inp}} = \mathbf{Z} \cdot \mathbf{R}_{\mathsf{inp}} + x^\top \otimes \mathbf{G}$ and decoding vector $\mathbf{u} = \mathbf{Z} \cdot \mathbf{r}_{\mathbf{u}}$ for $\mathbf{r}_{\mathbf{u}} \xleftarrow{R} \{0,1\}^m$.

- Hyb_4: In this experiment, the challenger generates $\mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{B}}, \mathbf{C}_x$ and dk from $\mathbf{C}_{\mathbf{Z}}$.

  3.(c) Set garbled input label $L_x = (\mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{Z}}, \mathbf{C}_{\mathbf{B}}, \mathbf{C}_x, \mathbf{B}_{\mathsf{inp}})$ where

$$\mathbf{C}_{\mathbf{Z}} = \mathbf{s}^\top \cdot \mathbf{Z} + \mathbf{e}_0^\top \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{A}}$$

$$\mathbf{C}_{\mathbf{B}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{B}} \quad , \quad \mathbf{C}_x = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_x.$$

  3.(d) Set decoding key $\mathsf{dk} = \left\lfloor \left((\mathbf{C}_{\mathsf{out}} + \mathbf{e}_{\mathsf{large}}) \cdot \mathbf{G}^{-1}(\mathbf{u}) - \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{r}_{\mathbf{u}} \cdot C(x)^\top \right) + \delta \right\rceil_p$.

- Hyb_5: In this experiment, $\mathbf{C}_{\mathbf{Z}}$ is replaced with a uniform random vector.

  3.(c) Set garbled input label $L_x = (\mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{Z}}, \mathbf{C}_{\mathbf{B}}, \mathbf{C}_x, \mathbf{B}_{\mathsf{inp}})$ where

$$\mathbf{C}_{\mathbf{Z}} \xleftarrow{R} \mathbb{Z}_q^t \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{A}}$$

$$\mathbf{C}_{\mathbf{B}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{B}} \quad , \quad \mathbf{C}_x = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_x.$$

Lemma 7.15. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathsf{Hyb}_{\mathsf{real}}(\mathcal{A}, \lambda) = 1] \leq \Pr[\mathsf{Hyb}_0(\mathcal{A}, \lambda) = 1] - \mathsf{negl}(\lambda).$$

Proof. Observe when the first m entries of $\mathbf{s}'$ are in fact $\lfloor \mathbf{s}^\top \cdot \mathbf{G} \rceil_p$, the computation $\mathbf{s}^\top \cdot \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{I} \cdot p \cdot 2^{i-1}) \bmod p \equiv \mathbf{s}^\top \cdot 2^{i-1} \bmod p$ indeed recovers the ith bit of $\mathbf{s}$ when rounded to $\mathbb{Z}_2$. By construction, $\mathbf{s}^\top \mathbf{B}_{\mathbf{Z}} - \mathbf{C}_{\mathbf{Z}} = \mathbf{e}_0^\top$, which is bounded entries $||\mathbf{e}_0^\top|| \leq \sigma$. □

Lemma 7.16. *For all adversaries $\mathcal{A}$,*

$$\Pr[\mathsf{Hyb}_0(\mathcal{A}, \lambda) = 1] = \Pr[\mathsf{Hyb}_{1,0}(\mathcal{A}, \lambda) = 1].$$

Proof. These hybrids are identical, as there is no $\mathsf{rk}^{(0)}$. □

Lemma 7.17. *For all adversaries $\mathcal{A}$ and $i \in [d]$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left| \Pr[\mathsf{Hyb}_{1,i-1}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Hyb}_{1,i}(\mathcal{A}, \lambda) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from preimage distribution property of lattice gadget trapdoor. □

Lemma 7.18. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left| \Pr[\mathsf{Hyb}_{1,d}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}, \lambda) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from the trapgen uniformity of lattice gadget trapdoors. □

Lemma 7.19. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left| \Pr[\mathsf{Hyb}_2(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}, \lambda) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from the leftover hash lemma (specifically the LHL compatibility of the key-homomorphic RAM encoding). □

Lemma 7.20. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}_3(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Hyb}_4(\mathcal{A},\lambda)=1]\right|=\mathsf{negl}(\lambda).$$

Proof. First observe that for $\mathbf{C_A}, \mathbf{C_B}$, and $\mathbf{C}_x$, this change is purely syntactic, as we can rewrite

$$\begin{aligned}
\mathbf{C_A} &= \mathbf{C_Z}\cdot\mathbf{R_A} = (\mathbf{s}^\intercal\cdot\mathbf{Z}+\mathbf{e}_0^\intercal)\cdot\mathbf{R_A} = \mathbf{s}^\intercal\cdot(\mathbf{Z}\cdot\mathbf{R_A})+\mathbf{e}_0^\intercal\cdot\mathbf{R_A} = \mathbf{s}^\intercal\cdot\mathbf{A}+\mathbf{e}_1^\intercal\\
\mathbf{C_B} &= \mathbf{C_Z}\cdot\mathbf{R_B} = (\mathbf{s}^\intercal\cdot\mathbf{Z}+\mathbf{e}_0^\intercal)\cdot\mathbf{R_B} = \mathbf{s}^\intercal\cdot((\mathbf{Z}\cdot\mathbf{R_B}+\mathsf{RHash}(0)\otimes\mathbf{G})-\mathsf{RHash}(0)\otimes\mathbf{G})+\mathbf{e}_0^\intercal\cdot\mathbf{R_B}\\
&= \mathbf{s}^\intercal\cdot(\mathbf{B}-\mathsf{RHash}(0)\otimes\mathbf{G})+\mathbf{e}_2^\intercal\\
\mathbf{C}_x &= \mathbf{C_Z}\cdot\mathbf{R}_x = (\mathbf{s}^\intercal\cdot\mathbf{Z}+\mathbf{e}_0^\intercal)\cdot\mathbf{R}_x = \mathbf{s}^\intercal\cdot((\mathbf{Z}\cdot\mathbf{R}_x+x^\intercal\otimes\mathbf{G})-x^\intercal\otimes\mathbf{G})+\mathbf{e}_0^\intercal\cdot\mathbf{R}_x\\
&= \mathbf{s}^\intercal\cdot(\mathbf{B}_{\mathsf{inp}}-x^\intercal\otimes\mathbf{G})+\mathbf{e}_3^\intercal
\end{aligned}$$

For the decoding key, first we can observe similar to the prior cases that $\mathbf{C_Z}\cdot\mathbf{r_u} = \mathbf{s}^\intercal\cdot\mathbf{u}+\mathbf{e}_0^\intercal\cdot\mathbf{r_u}$. In addition, recall from our correctness analysis that $||\mathbf{C}_{\mathsf{out}}-\mathbf{s}^\intercal\cdot(\mathbf{B}_{\mathsf{out}}-C(x)\cdot\mathbf{G})|| \le B^{d+2}\cdot\sigma$. Taking these two together, we can bound

$$\left\|\left((\mathbf{C}_{\mathsf{out}}+\mathbf{e}_{\mathsf{large}}^\intercal)\cdot\mathbf{G}^{-1}(\mathbf{u})-\mathbf{C_Z}\cdot\mathbf{r_u}\cdot C(x)\right)-(\mathbf{s}^\intercal\cdot\mathbf{B}_{\mathsf{out}})\right\| \le B^{d+2}\cdot\sigma+\mathsf{poly}(\lambda).$$

Since we chose $\mathbf{e}_{\mathsf{large}} \gg (B^{d+2}\cdot\sigma)$, the noisy distributions are statistically close by a smudging argument Lemma 7.2. □

Lemma 7.21. *Assuming decomposed LWE, for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}_4(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Hyb}_5(\mathcal{A},\lambda)=1]\right|=\mathsf{negl}(\lambda).$$

Proof. Specifically, this will follow from the LWE indistinguishability property of the key-homomorphic RAM encoding, where

$$\{\mathsf{pp},\mathbf{Z},\mathbf{s}^\intercal\cdot\mathbf{Z}+\mathbf{e}^\intercal\}\approx\{\mathsf{pp},\mathbf{Z},\mathbf{t}^\intercal\}.$$

Observe at by Hyb_4, all the elements in the view of the adversary dependent on $\mathbf{s}$ can be derived from $\mathbf{C_Z} = \mathbf{s}^\intercal\cdot\mathbf{Z}+\mathbf{e}_0^\intercal$. This can be directly replaced with the (decomposed) LWE challenge vector. □

Lemma 7.22. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathsf{Hyb}_5(\mathcal{A},\lambda)=1]=\mathsf{negl}(\lambda).$$

Proof. Since $\mathbf{C_Z}$ is a random vector, with overwhelming probability there does not exist $\mathbf{s}_q$ for which $||\mathbf{C_Z}-\mathbf{s}_q\cdot\mathbf{Z}|| \le \sigma$. □

Taking together Lemmas 7.15 to 7.22 gives us the proof of our theorem. □

8 Reusable Fully Succinct Garbling

In this section, we build upon the prior construction of Section 7 to the setting of *private* garbling. Formally, we show the following theorem:

Theorem 8.1 (Reusable Succinct Private Garbling)**.** *Assuming decomposed LWE Assumption 7.5 parameterized as Construction 8.4, there exists a reusable succinct private garbling scheme* Π_{garble} *satisfying the following:*

- *For a circuit $C:\{0,1\}^\ell\to\{0,1\}$ of depth d, the public data* pd *and garbling material* $|\hat{C}|,|\hat{x}|,|\mathsf{gk}|$ *are of size* $\mathsf{poly}(\lambda,d)$ *(independent of circuit size $|C|$).*
- *The garbling algorithms* $(\mathsf{Setup},\mathsf{GarbC},\mathsf{GarbInp})$ *are computable in* NC^1.
- Π_{garble} *satisfies selective input-hiding simulation security.*

8.1 Preliminaries

We will reuse many of the preliminaries and notation from the prior Section 7. In addition, we introduce the following additional ingredients:

Definition 8.2 (Homomorphic Encryption and Evaluation à la [GSW13])**.** Let λ be a security parameter and n, m, q, σ be lattice parameters for $m \geq 2n \log q$. Let $C_{\ell,d}$ be a family of depth d boolean circuits $C : \{0,1\}^\ell \to \{0,1\}$. There exists efficient algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{pk}, \mathsf{sk}$: On input security parameter λ, the key generation algorithm outputs public and secret keys of the form:

$$\mathsf{pk} = \mathbf{B}_{\mathsf{fhe}} = \begin{bmatrix} \overline{\mathbf{B}}_{\mathsf{fhe}} \\ \mathbf{s}^\intercal \cdot \overline{\mathbf{B}}_{\mathsf{fhe}} + \mathbf{e}_{\mathsf{fhe}}^\intercal \end{bmatrix} \in \mathbb{Z}_q^{(n+1)\times m} \quad , \quad \mathsf{sk} = \mathbf{r}^\intercal = [\mathbf{s}^\intercal \mid -1] \in \mathbb{Z}_q^{n+1},$$

  where $\mathbf{s} \in \mathbb{Z}_q^n, \overline{\mathbf{B}}_{\mathsf{fhe}} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{e}_{\mathsf{fhe}} \xleftarrow{R} \mathcal{D}_\sigma$. We will in general use $\overline{\mathbf{B}}$ and $\underline{\mathbf{B}}$ to denote the first n rows and last row of matrices (either public keys or ciphertexts) $\mathbf{B} \in \mathbb{Z}_q^{(n+1)\times m}$.

- $\mathsf{Encrypt}(\mathsf{pk}, x) \to \mathsf{ct}$: On input public key $\mathsf{pk} = \mathbf{B}_{\mathsf{fhe}}$ and bit $x \in \{0,1\}^\ell$, the encryption algorithm outputs a ciphertext of the form:

$$\mathsf{ct}_x = \mathbf{B}_{\mathsf{fhe}}\mathbf{R} - x \otimes \mathbf{G} \in \mathbb{Z}_q^{(n+1)\times \ell \cdot m},$$

  where $\mathbf{R} \xleftarrow{R} \{0,1\}^{m \times \ell \cdot m}$ is the encryption randomness. Note that ct_x is decomposable, and can be written as $[\mathbf{B}_{\mathsf{fhe}} \cdot \mathbf{R}_1 - x_1 \cdot \mathbf{G} \mid \cdots \mid \mathbf{B}_{\mathsf{fhe}} \cdot \mathbf{R}_\ell - x_\ell \cdot \mathbf{G}]$ where $\mathbf{R} = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell]$.

- $\mathsf{FHEEval}(C) \to C_{\mathsf{fhe}}$: On input circuit $C \in C_{\ell,d}$, the homomorphic evaluation algorithm deterministically produces circuit $C_{\mathsf{fhe}} : \{0,1\}^{\ell \cdot (n+1) \cdot m \cdot \log q} \to \mathbb{Z}_q^{(n+1)\times m}$, which takes in a (bit representation) of ct_x and outputs a matrix $\mathbf{B}_{C(x)} \in \mathbb{Z}_q^{(n+1)\times m}$.

Since we will not be directly relying on the structure of the FHE scheme, we omit the security requirements associated with the above FHE scheme and describe the structural properties:

- **Homomorphism:** Let $\mathbf{B}_{\mathsf{fhe}}$ be an public key matrix and ciphertext $\mathsf{ct} = \mathsf{Bits}(\mathbf{X})$ for $\mathbf{X} = \mathbf{B}_{\mathsf{fhe}} \cdot \mathbf{R} - x^\intercal \otimes \mathbf{G}$ be an FHE ciphertext. Then there exists matrix $\mathbf{R}_C$ where

$$C_{\mathsf{fhe}}(\mathbf{X}) = \mathbf{B}_{\mathsf{fhe}} \cdot \mathbf{R}_C - C(x) \cdot \mathbf{G}$$

- **Low Norm:** In the above described relation, matrix $||\mathbf{R}_C|| \leq m^{O(d)} \cdot ||\mathbf{R}||$.

- **Circuit Depth:** For a circuit C of depth d, circuit C_{fhe} has $d \cdot O(\log m + \log\log q)$.

Definition 8.3 (Bit Packing and Homomorphic Evaluation for Matrix-Valued Circuits [BTVW17])**.** Let λ be a security parameter and n, m, q, σ be lattice parameters where $m \geq 2n \log q$ and $n \geq \lambda$. For matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m}$, let $\mathsf{Bits}(\mathbf{M}) \in \{0,1\}^{n \cdot m \cdot \log q}$ denote the entry-wise bit-decomposition and flattening of $\mathbf{M}$. Then there exists efficient (deterministic) algorithms:

- $\mathsf{Pack}(\mathbf{B}) \to \mathbf{B}_{\mathsf{Pack}}$: On input matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times(m^2 \cdot n \cdot \log q)}$, Pack computes matrix $\mathbf{B}_{\mathsf{Pack}} \in \mathbb{Z}_q^{n\times m}$.

- $\mathsf{PackE}(1^n, 1^m, q) \to \mathbf{H}_{\mathsf{Pack}}$: On input dimensions n, m, q, PackE produces low norm matrix $\mathbf{H}_{\mathsf{Pack}} \in \mathbb{Z}_q^{(m^2 \cdot n \cdot \log q)\times m}$

Such that the following properties hold:

- **Low Norm:** The matrix $\mathbf{H}_{\mathsf{Pack}}$ has low norm $||\mathbf{H}_{\mathsf{Pack}}|| = 1$.

- **Homomorphism:** For any matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times(n \cdot m^2 \cdot \log q)}$, the relation

$$(\mathbf{B} - \mathsf{Bits}(\mathbf{M})^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathsf{Pack}} = \mathbf{B}_{\mathsf{Pack}} - \mathbf{M}$$

  holds.

By composing this with Definition 7.8 encodings, this allows one to extend homomorphic evaluation to *matrix-valued* circuits, where the output of C takes the form of a matrix $\in \mathbb{Z}_q^{n\times m}$.

8.2 Construction

Our construction of private garbling follows from 'dual-use' technique of [BTVW17], used to lift attribute-based encryption to predicate encryption [GVW15, BTVW17] and attribute-based LFE to LFE [QWW18]. At a high level, this technique shares a decryption secret $\mathbf{s}$ between a GSW-style FHE scheme and the underlying base primitive (in our case the privacy-free garbling scheme). This enables input/attribute hiding by simply encrypting the input under the FHE scheme, and instead issuing garbled labels of the FHE ciphertexts, along with running the FHE evaluation of the circuit to be garbled. Then, using the structural similarity of the base primitive with the FHE scheme allows one to 'simultaneously' decode the attribute-based primitive while decrypting the FHE scheme.

Construction 8.4 (Succinct Reusable Garbling)**.** Let λ be the security parameter and d be a bound on circuit depth. Our scheme relies on the following lattice ingredients:

- GSW Fully homomorphic encryption Π_{fhe} Definition 8.2.
- Lattice homomorphic evaluation Definition 7.8.
- Bit-packing Definition 8.3.
- The privacy-free reusable garbling protocol $\Pi_{\mathsf{PF-Garble}}$ of Section 7, which we will use as-is except for the explicit specification of secret vector $\mathbf{s}$ and the decoding phase.

We set our parameters as follows:

$$n \geq \lambda \quad , \quad m = 3n \log q \quad , \quad B = \lambda^{\log(\lambda)} \quad , \quad \sigma_{\mathsf{large}} = \sigma \cdot B^{d_{\mathsf{fhe}}+3}$$
$$\Delta = B^{d_{\mathsf{fhe}}+4} \cdot \sigma \quad , \quad p = \sigma \cdot B \quad , \quad q = \Delta \cdot p,$$

where the $d_{\mathsf{fhe}} \in d \cdot O(\log m + \log\log q)$ is the depth of the FHE evaluation circuit C'_{fhe} specified below and $\mathsf{dLWE}_{n,m,q,2m^2\lambda,\chi,\sigma}$ is hard with respect to security parameter λ, which in turn fix parameters t, k of the RAM encoding. Our private garbling scheme proceeds as follows:

- **Input Encoding Phase:** The garbler samples public data $\mathsf{pd} \leftarrow \Pi_{\mathsf{PF-Garble}}.\mathsf{Setup}(1^\lambda, 1^{d_{\mathsf{fhe}}})$ of the privacy free garbling scheme, which will also act as public data of the private garbling scheme.

  To compute the input encoding, the garbler also samples FHE key-pair

  $$\mathsf{pk} = \mathbf{B}_{\mathsf{fhe}} = \begin{bmatrix} \overline{\mathbf{B}}_{\mathsf{fhe}} \\ \mathbf{s}^\intercal \cdot \overline{\mathbf{B}}_{\mathsf{fhe}} + \mathbf{e}^\intercal_{\mathsf{fhe}} \end{bmatrix} \in \mathbb{Z}_q^{(n+1)\times m} \quad , \quad \mathsf{sk} = \mathbf{r}^\intercal = [\mathbf{s}^\intercal \mid -1] \in \mathbb{Z}_q^{n+1},$$

  where $\mathbf{B}_{\mathsf{fhe}} \xleftarrow{R} \mathbb{Z}_q^{n\times m}, \mathbf{s}^\intercal \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{e}^\intercal_{\mathsf{fhe}} \xleftarrow{R} \mathcal{D}_\sigma^m$. Using this key, the garbler computes ciphertext $\mathsf{ct}_x \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x)$, and derives privacy-free input encoding $[\![\mathsf{ct}_x]\!] \leftarrow \mathsf{Enc}(\mathsf{gk}, \mathsf{Bits}(\mathsf{ct}_x))$ using the same secret $\mathbf{s}$. The garbler sends public data $\mathsf{pd}, \mathsf{pk}, [\![\mathsf{ct}_x]\!]$ to the evaluator.

- **Garbling:** Define the circuit $C'_{\mathsf{fhe}} = \mathsf{Bits}(\overline{\mathsf{FHEEval}(C)})$ - in other words, outputs the bit representation of the first n rows of the evaluation of C under homomorphic encryption. The garbler and evaluator run $\Pi_{\mathsf{PF-Garble}}$ on circuit C'_{fhe} and input (encoding) $[\![\mathsf{Bits}(\mathsf{ct}_x)]\!]$.

- **Output Phase:** Let $\mathbf{X}_{\mathsf{out}} \in \mathbb{Z}_q^{n+1\times m}$ be the homomorphic evaluation of $\mathsf{FHEEval}(C)$ on input ct_x. The privacy-free garbling will produce encodings $[\![\mathsf{Bits}(\overline{\mathbf{X}}_{\mathsf{out}})]\!]$ which contain shares of the form:

  $$[\![\mathsf{Bits}(\overline{\mathbf{X}}_{\mathsf{out}})]\!]_G = \mathbf{B}_{\mathsf{out}} \in \mathbb{Z}_q^{n\times(m^2\cdot n\cdot\log q)} \quad , \quad [\![\mathsf{Bits}(\overline{\mathbf{X}}_{\mathsf{out}})]\!]_E = (\mathsf{Bits}(\overline{\mathbf{X}}_{\mathsf{out}}), \mathbf{B}_{\mathsf{out}}, \mathbf{C}_{\mathsf{out}})$$

  - The garbler outputs authentication key as $\Pi_{\mathsf{PF-Garble}}$, but with final entry decoding key as

    $$\mathsf{dk} = \left\lfloor (\mathbf{s}^\intercal \cdot (\mathbf{B}_{\mathsf{Pack}} - \overline{\mathbf{B}}_{\mathsf{fhe}} \cdot \mathbf{R}_C) + \mathbf{e}^\intercal_{\mathsf{large}}) \cdot \mathbf{G}^{-1}(\mathbf{u}) + \delta \right\rceil_p$$

    where $\mathbf{B}_{\mathsf{Pack}} \leftarrow \mathsf{Pack}(\mathbf{B}_{\mathsf{out}})$ and $\mathbf{R}_C$ is the matrix corresponding to FHE homomorphism relation.
  - The evaluator computes all but the last entry of the output label as $\Pi_{\mathsf{PF-Garble}}$, which is computed as

    $$\left\lfloor \mathbf{C}_{\mathsf{out}} \cdot \mathbf{H}_{\mathsf{Pack}} \cdot \mathbf{G}^{-1}(\mathbf{u}) + \delta \right\rceil_p$$

    where $\mathbf{H}_{\mathsf{Pack}}$ is computed through $\mathbf{H}_{\mathsf{Pack}} \leftarrow \mathsf{PackE}(1^n, 1^m, q)$.

8.3 Garbler Depth Complexity

We can separate the garbler's computation into two components:

- **FHE-related Computation**: The garbler computes the FHE KeyGen, Encrypt, FHEEval functions. Similar to Section 7.3, we simply observe these can all be implemented from elementary arithmetic/matrix operations over a poly-size modulus, which can be computed in NC^1. This includes the modified decoding phase.
- **Privacy-free Garbling Computation**: Since the circuit and inputs of our privacy-free garbling are polynomial, this follows directly from the NC^1 depth of $\Pi_{\mathsf{PF-Garble}}$.

8.4 Succinctness and Decomposability

Observe the garbling material consists of garbling material for $\Pi_{\mathsf{PF-Garble}}$ run on FHE ciphertext input bits ct_x and FHE homomorphic evaluation circuit C'_{fhe}, as well as the FHE ciphertext and public key. We can observe directly from our parameter choices that the FHE public key and ciphertext are of size $\mathsf{poly}(\lambda, d)$. From the succinctness of $\Pi_{\mathsf{PF-Garble}}$, the garbling material for $\Pi_{\mathsf{PF-Garble}}$ is of size $\mathsf{poly}(\lambda, d_{\mathsf{fhe}})$, which we recall from the properties of our FHE scheme is $\mathsf{poly}(\lambda, d \cdot O(\log m + \log\log q))$. From inspection, we can see parameters $m, \log q$ are both $m, \log q$ are both $\mathsf{poly}(\lambda, d)$, which gives our the garbling material for $\Pi_{\mathsf{PF-Garble}}$ to be $\mathsf{poly}(\lambda, d)$-succinct.

Note that the labels of our scheme correspond to labels of bits of the FHE ciphertext under the privacy-free garbling scheme. Thus decomposability follows directly from decomposability of the FHE scheme and $\Pi_{\mathsf{PF-Garble}}$.

8.5 Correctness

To show correctness of our scheme, it suffices to show if conforms to our notion of algebraic labeling. Since all but the last term of the evaluator's label are directly generated as the privacy-free garbling label, the correctness of these follow directly from the correctness of the privacy-free scheme. For the decoding key/final entry of the authentication key, recall from the analysis of privacy-free garbling that we can bound the privacy-free output labels

$$||\mathbf{C}_{\mathsf{out}} - \mathbf{s}^\intercal(\mathbf{B}_{\mathsf{out}} - \mathsf{Bits}(\overline{\mathbf{X}}_{\mathsf{out}}) \otimes \mathbf{G})|| \leq B^{d_{\mathsf{fhe}}+2} \cdot \sigma$$

We can rewrite this as

$$\begin{aligned}
\mathbf{C}_{\mathsf{out}} &= \mathbf{s}^\intercal(\mathbf{B}_{\mathsf{out}} - \mathsf{Bits}(\overline{\mathbf{X}}_{\mathsf{out}}) \otimes \mathbf{G}) + \mathbf{e}^\intercal \\
\Leftrightarrow \mathbf{C}_{\mathsf{out}} \cdot \mathbf{H}_{\mathsf{Pack}} &= \mathbf{s}^\intercal \cdot (\mathbf{B}_{\mathsf{Pack}} - \overline{\mathbf{X}}_{\mathsf{out}}) + \mathbf{e}^\intercal \cdot \mathbf{H}_{\mathsf{Pack}} \\
&= \mathbf{s}^\intercal \cdot (\mathbf{B}_{\mathsf{Pack}} - \overline{\mathbf{B}}_{\mathsf{fhe}} \cdot \mathbf{R}_C - C(x) \cdot \mathbf{G}) + \mathbf{e}^\intercal \cdot \mathbf{H}_{\mathsf{Pack}} \\
\Leftrightarrow \mathbf{C}_{\mathsf{out}} \cdot \mathbf{H}_{\mathsf{Pack}} \cdot \mathbf{G}^{-1}(\mathbf{u}) &= \mathbf{s}^\intercal \cdot (\mathbf{B}_{\mathsf{Pack}} \cdot \mathbf{G}^{-1}(\mathbf{u}) - \overline{\mathbf{B}}_{\mathsf{fhe}} \cdot \mathbf{R}_C \cdot \mathbf{G}^{-1}(\mathbf{u}) - C(x) \cdot \mathbf{u}) + \mathbf{e}^\intercal \cdot \mathbf{H}_{\mathsf{Pack}} \cdot \mathbf{G}^{-1}(\mathbf{u})
\end{aligned}$$

This tells us the difference between the last label entry and decoding key is once again $\mathbf{s}^\intercal \cdot C(x) \cdot \mathbf{u}$ along with error bounded by $||\mathbf{e}^\intercal \cdot \mathbf{H}_{\mathsf{Pack}} \cdot \mathbf{G}^{-1}(\mathbf{u})|| \leq B^{d_{\mathsf{fhe}}+2} \cdot \sigma \cdot \mathsf{poly}(\lambda)$ (as $\mathbf{H}_{\mathsf{Pack}}, \mathbf{G}^{-1}(\mathbf{u})$ are both binary matrices). Thus, we can observe again that because Δ is superpolynomially larger than this error, our scheme is correct with overwhelming probability.

8.6 Security

Analogous to the prior section, we will prove the semantic security of of Construction 8.4, which by Lemma 3.4 suffices to show input-hiding simulation security.

Theorem 8.5 (Semantic Security)**.** *Assuming decomposed LWE, garbling scheme Construction 8.4 satisfies semantic security as an algebraic garbling scheme.*

Proof. The security proof follows a fairly similar structure to Theorem 7.14, where we incrementally remove trapdoors and sample all matrices with respect to $\mathbf{Z}, \mathbf{C}_{\mathbf{Z}}$, which allows us to replace the latter with a uniform random vector through decomposed LWE. Consider the following sequence of hybrids:

- $\mathsf{Hyb}^{(b)}_{\mathsf{real}}$: This is the real selective semantic security experiment instantiated with Construction 8.4. We highlight the instances this differs from Theorem 7.14 (primarily where FHE is used).

  1. The adversary $\mathcal{A}$ specifies circuit C and inputs $x_0, x_1 \in \{0,1\}^\ell$ to the challenger.
  2. The challenger samples the following:
     (a) Setup material $(\mathsf{pp}, \mathbf{Z}) \leftarrow \mathsf{RSetup}(1^\lambda)$.
     (b) Randomizing binary matrices:

$$\mathbf{R}_{\mathbf{A}} \xleftarrow{R} \{0,1\}^{t\times m}, \mathbf{R}_{\mathbf{B}} \xleftarrow{R} \{0,1\}^{t\times m\cdot k}, \mathbf{R}_{\mathsf{inp}} \xleftarrow{R} \{0,1\}^{t\times m\cdot \ell}, \mathbf{R}_{\mathsf{fhe}} \leftarrow \{0,1\}^{m\times \ell\cdot m}, \mathbf{R}_{\mathsf{enc}} \xleftarrow{R} \{0,1\}^{t\times m}$$

     (c) Trapdoor matrix $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
     (d) Encoding matrices $\mathbf{B} \xleftarrow{R} \mathbb{Z}_q^{n\times k\cdot m}$, $\mathbf{B}_{\mathsf{inp}} \xleftarrow{R} \mathbb{Z}_q^{n\times m\cdot \ell}$, FHE matrix $\overline{\mathbf{B}}_{\mathsf{fhe}} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ and decoding vector $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$.
     (e) LWE secret and error vectors :

$$\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n \quad , \quad \mathbf{e}_0 \xleftarrow{R} \mathcal{D}_\sigma^t, \quad , \quad \mathbf{e}_{\mathsf{large}} \xleftarrow{R} \mathcal{D}_{\sigma_{\mathsf{large}}}^{m\cdot k}$$

$$\mathbf{e}_1^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_{\mathbf{A}} \quad , \quad \mathbf{e}_2^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_{\mathbf{B}} \quad , \quad \mathbf{e}_3^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_{\mathsf{inp}} \quad , \quad \mathbf{e}_{\mathsf{fhe}}^\intercal = \mathbf{e}_0^\intercal \cdot \mathbf{R}_{\mathsf{enc}}.$$

     (f) Rekeying material for each $i \in [d_{\mathsf{fhe}}]$:

$$\mathbf{B}^{(0)} \leftarrow \mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, [\ell], \mathbf{B}_{\mathsf{inp}}) \quad , \quad \mathbf{B}^{(i)} \xleftarrow{R} \mathbb{Z}_q^{n\times k\cdot m}$$

$$\mathbf{B}_{C_i}^{(i-1)} \leftarrow \mathsf{SEvalF}(\mathsf{pp}, [\mathbf{B}^{(0)} \mid \ldots \mid \mathbf{B}^{(i-1)}], C_i) \quad , \quad \mathsf{rk}^{(i)} \leftarrow \mathsf{SamplePre}\left([\mathbf{A} \mid \mathbf{Z}], \begin{bmatrix} \mathsf{td}_{\mathbf{A}} \\ 0 \end{bmatrix}, \mathbf{B}^{(i)} - \mathbf{B}_{C_i}^{(i-1)}, 1\right)$$

     (g) FHE public key and ciphertext:

$$\mathbf{B}_{\mathsf{fhe}} = \begin{bmatrix} \overline{\mathbf{B}}_{\mathsf{fhe}} \\ \mathbf{s}^\intercal \overline{\mathbf{B}}_{\mathsf{fhe}} + \mathbf{e}_{\mathsf{fhe}}^\intercal \end{bmatrix} \quad , \quad \mathbf{X} = \mathbf{B}_{\mathsf{fhe}} \cdot \mathbf{R}_{\mathsf{fhe}} - x_b \otimes \mathbf{G}$$

  3. The challenger computes the adversary's view as follows:
     (a) Set public data $\mathsf{pd} = (\mathsf{pp}, \mathbf{Z}, \mathbf{B})$.
     (b) Set garbled circuit and public garbling key $\hat{C} = \mathsf{gk} = \{\mathsf{rk}^{(i)}\}_{i\in[d]}$.
     (c) Set garbled input label $L_x = (\mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{Z}}, \mathbf{C}_{\mathbf{B}}, \mathbf{C}_x, \mathbf{B}_{\mathsf{inp}})$ where

$$\mathbf{C}_{\mathbf{Z}} = \mathbf{s}^\intercal \cdot \mathbf{Z} + \mathbf{e}_0^\intercal \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{s}^\intercal \cdot \mathbf{A} + \mathbf{e}_1^\intercal$$

$$\mathbf{C}_{\mathbf{B}} = \mathbf{s}^\intercal \cdot (\mathbf{B} - \mathsf{RHash}(0) \otimes \mathbf{G}) + \mathbf{e}_2^\intercal \quad , \quad \mathbf{C}_x = \mathbf{s}^\intercal \cdot (\mathbf{B}_{\mathsf{inp}} - \mathsf{Bits}(\mathbf{X}) \otimes \mathbf{G}) + \mathbf{e}_3^\intercal.$$

  4. The adversary outputs a bit b', which is the output of the experiment.

- $\mathsf{Hyb}^{(b)}_{1,i}$: In this experiment, the challenger samples the rekeying keys without using $\mathsf{td}_{\mathbf{A}}$:

  2.(f) Rekeying material for each $i \in [d]$:

$$\mathbf{B}^{(0)} \leftarrow \mathsf{RWrite}(\mathsf{pp}, \mathbf{B}, [\ell], \mathbf{B}_{\mathsf{inp}}) \quad , \quad \mathsf{rk}^{(i)} \xleftarrow{R} \mathcal{D}_B^{(m+t)\times k\cdot m}$$

$$\mathbf{B}_{C_i}^{(i-1)} \leftarrow \mathsf{SEvalF}(\mathsf{pp}, [\mathbf{B}^{(0)} \mid \ldots \mid \mathbf{B}^{(i-1)}], C_i) \quad , \quad \mathbf{B}^{(i)} = [\mathbf{A} \mid \mathbf{Z}] \cdot \mathsf{rk}^{(i)} + \mathbf{B}_{C_i}^{(i-1)}$$

- $\mathsf{Hyb}^{(b)}_2$: In this experiment, the challenger samples $\mathbf{A}$ as a random matrix.

  2.(c) ~~Trapdoor~~ matrix $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$.

- $\mathsf{Hyb}_3^{(b)}$: In this experiment, the challenger derives matrices $\mathbf{A}, \mathbf{B}, \mathbf{B}_{\mathsf{inp}}$ *and* $\mathbf{B}_{\mathsf{fhe}}$ from $\mathbf{Z}$.

  2.(c) Matrix $\mathbf{A} = \mathbf{Z} \cdot \mathbf{R}_{\mathbf{A}}$.

  2.(d) Encoding matrices $\mathbf{B} = \mathbf{Z} \cdot \mathbf{R}_{\mathbf{B}} + \mathsf{RHash}(0) \otimes \mathbf{G}$, $\mathbf{B}_{\mathsf{inp}} = \mathbf{Z} \cdot \mathbf{R}_{\mathsf{inp}} + \mathsf{Bits}(\mathbf{X}) \otimes \mathbf{G}$, FHE matrix $\overline{\mathbf{B}}_{\mathsf{fhe}} = \mathbf{Z} \cdot \mathbf{R}_{\mathsf{enc}}$ and decoding vector $\mathbf{u} = \mathbf{Z} \cdot \mathbf{r}_{\mathbf{u}}$ for $\mathbf{r}_{\mathbf{u}} \xleftarrow{R} \{0,1\}^m$.

- $\mathsf{Hyb}_4^{(b)}$: In this experiment, the challenger generates $\mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{B}}, \mathbf{C}_x$ and $\underline{\mathbf{B}}_{\mathsf{fhe}}$ from $\mathbf{C}_{\mathbf{Z}}$.

  2.(g) FHE public key and ciphertext:

$$\mathbf{B}_{\mathsf{fhe}} = \begin{bmatrix} \overline{\mathbf{B}}_{\mathsf{fhe}} \\ \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathsf{enc}} \end{bmatrix} \quad , \quad \mathbf{X} = \mathbf{B}_{\mathsf{fhe}} \cdot \mathbf{R}_{\mathsf{fhe}} - x_b \otimes \mathbf{G}$$

  3.(c) Set garbled input label $L_x = (\mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{Z}}, \mathbf{C}_{\mathbf{B}}, \mathbf{C}_x, \mathbf{B}_{\mathsf{inp}})$ where

$$\mathbf{C}_{\mathbf{Z}} = \mathbf{s}^{\mathsf{T}} \cdot \mathbf{Z} + \mathbf{e}_0^{\mathsf{T}} \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{A}}$$

$$\mathbf{C}_{\mathbf{B}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{B}} \quad , \quad \mathbf{C}_x = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_x.$$

- $\mathsf{Hyb}_5^{(b)}$: In this experiment, $\mathbf{C}_{\mathbf{Z}}$ is replaced with a uniform random vector.

  3.(c) Set garbled input label $L_x = (\mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{Z}}, \mathbf{C}_{\mathbf{B}}, \mathbf{C}_x, \mathbf{B}_{\mathsf{inp}})$ where

$$\mathbf{C}_{\mathbf{Z}} \xleftarrow{R} \mathbb{Z}_q^t \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{A}}$$

$$\mathbf{C}_{\mathbf{B}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{B}} \quad , \quad \mathbf{C}_x = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_x.$$

- $\mathsf{Hyb}_6^{(b)}$: In this experiment, the FHE public key $\mathbf{B}_{\mathsf{fhe}}$ is replaced with a uniform matrix.

  2.(d) Encoding matrices $\mathbf{B} = \mathbf{Z} \cdot \mathbf{R}_{\mathbf{B}} + \mathsf{RHash}(0) \otimes \mathbf{G}$, $\mathbf{B}_{\mathsf{inp}} = \mathbf{Z} \cdot \mathbf{R}_{\mathsf{inp}} + \mathsf{Bits}(\mathbf{X}) \otimes \mathbf{G}$, ~~FHE matrix $\overline{\mathbf{B}}_{\mathsf{fhe}} = \mathbf{Z} \cdot \mathbf{R}_{\mathsf{enc}}$~~ and decoding vector $\mathbf{u} = \mathbf{Z} \cdot \mathbf{r}_{\mathbf{u}}$ for $\mathbf{r}_{\mathbf{u}} \xleftarrow{R} \{0,1\}^m$.

  2.(g) FHE public key and ciphertext:

$$\mathbf{B}_{\mathsf{fhe}} \xleftarrow{R} \mathbb{Z}_q^{(n+1) \times m} \quad , \quad \mathbf{X} = \mathbf{B}_{\mathsf{fhe}} \cdot \mathbf{R}_{\mathsf{fhe}} - x_b \otimes \mathbf{G}$$

- $\mathsf{Hyb}_7^{(b)}$: In this experiment, the FHE ciphertext $\mathbf{X}$ is replaced with a uniform matrix.

  2.(g) FHE public key and ciphertext:

$$\mathbf{B}_{\mathsf{fhe}} \xleftarrow{R} \mathbb{Z}_q^{(n+1) \times m} \quad , \quad \mathbf{X} \xleftarrow{R} \mathbb{Z}_q^{(n+1) \times (m \cdot \ell)}$$

The proof of semantic security proceeds essentially identically to that of Theorem 7.14, where the additional FHE public key component $\overline{\mathbf{B}}_{\mathsf{fhe}}$ is simulated as just as matrices $\mathbf{A}, \mathbf{B}$, through $\mathbf{Z}$. The only difference is once $\mathbf{C}_{\mathbf{Z}}$ is replaced with uniform in $\mathsf{Hyb}_5^{(b)}$, we subsequently rely on LHL (with randomness $\mathbf{B}_{\mathsf{enc}}, \mathbf{B}_{\mathsf{fhe}}$) in $\mathsf{Hyb}_6^{(b)}$, $\mathsf{Hyb}_7^{(b)}$ to replace the FHE ciphertext with a random matrix, thus hiding the input. □

We note that a similar argument as above also suffices to show this *private* garbling scheme also satisfies authenticity/one-wayness.

9 Attribute-Based Encryption

Our techniques in Section 7 naturally extend to yield a key-policy attribute-based encryption scheme for circuits which enjoys succinct secret keys and NC^1 key generation. We note this in turn can be used to recover a (somewhat less direct) construction of reusable garbling via [GKP+13] in combination with any FHE and (single use) garbling scheme.

9.1 Definitions

We recall the definition of attribute-based encryption below:

Definition 9.1 (Key-Policy Attribute-Based Encryption)**.** Let λ be a security parameter, and $\mathcal{M} = \{\mathcal{M}_\lambda\}$ be a message space, and $\mathcal{F} = \mathcal{F}_{\lambda,z}$ be a class of functions $f : \{0,1\}^\ell \to \{0,1\}$ for some $\ell = \ell(\lambda, z)$. Then a key-policy ABE scheme Π_{ABE} is a tuple of efficient algorithms with the following syntax.

- $\mathsf{Setup}(1^\lambda, z) \to (\mathsf{mpk}, \mathsf{msk})$: On input security parameter λ and functionality index z, the setup algorithm outputs a master public key mpk and master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$: On input master secret key msk and function $f \in \mathcal{F}$, the key generation algorithm outputs a function secret key sk_f.
- $\mathsf{Encrypt}(\mathsf{mpk}, x, \mu) \to \mathsf{ct}_x$: On input master public key mpk, input $x \in \{0,1\}^\ell$, and message $\mu \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct_x.
- $\mathsf{Decrypt}((\mathsf{sk}_f, f), \mathsf{ct}_x) \to \mu \cup \bot$: On input function secret key sk_f and ciphertext ct_x, the decryption algorithm outputs message μ or $\bot$ indicating decryption failure. Since we are in the succinct setting where sk_f is potentially shorter than the description of f, the decryption algorithm will also explicitly get f.

Moreover, the algorithms should satisfy the following properties:

- **Correctness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all z, all messages $\mu \in \mathcal{M}$, functions $f \in \mathcal{F}$ and inputs $x \in \{0,1\}^\ell$ where $f(x) = 0$,

$$\Pr\left[\mu' = \mu : \begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, z) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \\ \mathsf{ct}_x \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, x, \mu) \\ \mu' \leftarrow \mathsf{Decrypt}((\mathsf{sk}_f, f), \mathsf{ct}_x)\end{array}\right] = \mathsf{negl}(\lambda)$$

- **(Selective) Security:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, define the following security game $\mathsf{ExptABE}_{\mathcal{A}}$ between $\mathcal{A}$ and a challenger:

  1. **Setup Phase:** The adversary $\mathcal{A}$ specifies an input x^* and functionality index z. The challenger computes $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, z)$ and returns mpk to the adversary.
  2. **Keygen Phase:** As many times as desired, the adversary $\mathcal{A}$ submits a function $f \in \mathcal{F}$ subject to $f(x) = 1$. The challenger replies with $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$.
  3. **Challenge Phase:** The adversary $\mathcal{A}$ submits two messages μ_0, μ_1. The challenger responds with $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, x^*, \mu_b)$.
  4. **(Post-Challenge) Keygen Phase:** The adversary $\mathcal{A}$ can once again submit functions $f \in \mathcal{F}$ subject to $f(x)$ and receive $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$.
  5. **Output Phase:** Adversary $\mathcal{A}$ submits a bit b', which is the output of the experiment.

  We say a scheme Π_{ABE} satisfies selective security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathsf{ExptABE}^{(1)}_{\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{ExptABE}^{(0)}_{\mathcal{A}}(\lambda) = 1\right]\right| = \mathsf{negl}(\lambda).$$

9.2 Construction

We present our construction of KP-ABE as follows:

Construction 9.2 (Key-Policy Attribute-Based Encryption for Circuits)**.** Let λ be the security parameter and d be a depth bound Our scheme relies on the same lattice ingredients as our construction of privacy-free garbling:

- A lattice gadget trapdoor sampler (TrapGen, SamplePre) per Definition 7.6.
- Lattice-based homomorphic evaluation (EvalF, EvalFX) per Definition 7.8.
- Key-homomorphic RAM encoding ((RSetup, RHash, RRead, RXRead, RWrite, RXWrite)) per Definition 7.9.

Similar to the analysis of Section 7, we will directly use the composition of lattice and RAM homomorphism (SEvalF, SEvalFX) as described in Remark 7.13. We parameterize our scheme as follows:

$$n \geq \lambda \quad , \quad m = 3n \log q \quad , \quad B = \lambda^{\log(\lambda)} \quad , \quad \sigma_i = B^i \text{ for } i \in [d] \quad , \quad \sigma_{\mathsf{out}} = B^{d+1} \quad , \quad q \geq B^{2d+2} \cdot \sigma$$

where the $\mathsf{dLWE}_{n,m,q,2m^2\lambda,\chi,\sigma}$ is hard with respect to security parameter λ, which in turn fix parameters t, k of the RAM encoding. We can now present our scheme as follows:

- $\mathsf{Setup}(1^\lambda, 1^d)$: On input security parameter λ and depth parameter d, the setup algorithm proceeds as follows:
  - Sample RAM encoding setup $(\mathsf{pp}, \mathbf{Z}) \leftarrow \mathsf{RSetup}(1^\lambda)$.
  - Sample trapdoored matrix $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
  - Sample matrix $\mathbf{B}^{(0)} \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$
  - Sample vector $\mathbf{p} \xleftarrow{R} \mathbb{Z}_q^n$.
  - Output $(\mathsf{mpk} = (\mathsf{pp}, \mathbf{Z}, \mathbf{A}, \mathbf{B}^{(0)}, \mathbf{p}), \mathsf{msk} = (\mathsf{td}_{\mathbf{A}}, \mathsf{mpk})$.
- $\mathsf{KeyGen}(\mathsf{msk}, C)$: On input master secret key $\mathsf{msk} = \mathsf{td}_{\mathbf{A}}$ and circuit C, define circuit C_i to be the depth 1 taking in all depth $\leq i-1$ wires and outputting all depth $\leq i$ wires[18], the keygen algorithm:
  - For $i \in [d]$, sample base matrices $\mathbf{B}^{(i)} \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$
  - For $i \in [d]$, set $\mathbf{B}_{C_i}^{(i-1)} \leftarrow \mathsf{SEvalF}(\mathsf{pp}, \mathbf{B}^{(i-1)}, C_i)$.
  - For $i \in [d]$, sample $\mathsf{rk}^{(i)} \leftarrow \mathsf{SamplePre}\left([\mathbf{A} \mid \mathbf{Z}], \begin{bmatrix} \mathsf{td}_{\mathbf{A}} \\ \mathbf{0} \end{bmatrix}, \mathbf{B}_{C_i}^{(i-1)} - \mathbf{B}^{(i)}, \sigma_i\right)$.
  - Set $\mathbf{B}_{\mathsf{out}}^{(d)} \leftarrow \mathsf{RRead}(\mathsf{pp}, \mathbf{B}^{(d)}, |C|)$.
  - Sample decryption component $\mathbf{k}_C \leftarrow \mathsf{SamplePre}\left([\mathbf{A} \mid \mathbf{Z} \mid \mathbf{B}_{\mathsf{out}}^{(d)}], \begin{bmatrix} \mathsf{td}_{\mathbf{A}} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \mathbf{p}, \sigma_{\mathsf{out}}\right)$.
  - Return $\mathsf{sk}_C = (\{\mathsf{rk}^{(i)}\}_{i \in [d]}, \mathbf{k}_C, \mathsf{mpk})$.
- $\mathsf{Encrypt}(\mathsf{mpk}, x, \mu)$: On input master public key mpk, attribute set x and message $\mu \in \{0,1\}$, the encryption algorithm computes the following:
  - Sample encryption randomness $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^m$.
  - Sample noise vectors $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ as:

$$\mathbf{e}_0 \xleftarrow{R} \mathcal{D}_\sigma^t \quad , \quad \mathbf{R}_{\mathbf{A}} \xleftarrow{R} \{0,1\}^{t \times m} \quad , \quad \mathbf{R}_{\mathbf{B}} \xleftarrow{R} \{0,1\}^{m+t \times m \cdot k} \quad , \quad \mathbf{R}_{\mathbf{p}} \xleftarrow{R} \{0,1\}^{t \times 1}$$

$$\mathbf{e}_1^\top = \mathbf{e}_0^\top \cdot \mathbf{R}_{\mathbf{A}} \quad , \quad \mathbf{e}_2^\top = [\mathbf{e}_1 \mid \mathbf{e}_0^\top] \cdot \mathbf{R}_{\mathbf{B}} \quad , \quad \mathbf{e}_3^\top = \mathbf{e}_0^\top \cdot \mathbf{R}_{\mathbf{p}}.$$

  - Set ciphertext components

$$\mathbf{C}_{\mathbf{Z}} = \mathbf{s}^\top \mathbf{Z} + \mathbf{e}_0^\top \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{s}^\top \mathbf{A} + \mathbf{e}_1^\top$$

$$\mathbf{C}_x = \mathbf{s}^\top (\mathbf{B} - \mathsf{RHash}(x) \otimes \mathbf{G}) + \mathbf{e}_2^\top \quad , \quad \mathbf{C}_\mu = \mathbf{s}^\top \mathbf{p} + \mathbf{e}_3^\top + \mu \cdot \left\lceil \frac{q}{2} \right\rfloor.$$

  - Output ciphertext $\mathsf{ct} = (x, \mathbf{C}_{\mathbf{A}}, \mathbf{C}_{\mathbf{Z}}, \mathbf{C}_x, \mathbf{C}_\mu)$.

[18] Note this departs slightly different from the notation in prior sections, where C_i only computes depth exactly i wires

- $\mathsf{Decrypt}(C, \mathsf{sk}_C, \mathsf{ct}_x)$: On input secret key sk and ciphertext ct, the decryption algorithm Decrypt
  - Parse $\mathsf{sk}_C = (\{\mathsf{rk}^{(i)}\}_{i\in[d]}, \mathbf{k}_C, \mathsf{mpk})$ and $\mathsf{ct} = (x, \mathbf{C_A}, \mathbf{C_Z}, \mathbf{C}_x, \mathbf{C}_\mu)$ where $\mathsf{mpk} = (\mathsf{pp}, \mathbf{Z}, \mathbf{A}, \mathbf{B}^{(0)}, \mathbf{p})$.
  - Set $h^{(0)} = \mathsf{RHash}(x)$ and $\mathbf{C}^{(0)} = \mathbf{C}_x$.
  - For $i \in [d]$, let $x^{(i)}$ be all of the layer $\leq i$ wire values of $C(x)$ and let $h^{(i)} = \mathsf{RHash}(x^{(i)})$. Compute $(\mathbf{B}^{(i)}, \mathbf{C}^{(i)})$ as follows:
    * Compute homomorphic evaluation

$$\mathbf{B}^{(i-1)}_{C_i} = \mathsf{SEvalF}(\mathsf{pp}, \mathbf{B}^{(i-1)}, C_i) \quad , \quad \mathbf{C}^{(i-1)}_{C_i} = [\mathbf{C}^{(i-1)} \mid \mathbf{C_Z}] \cdot \mathbf{H}_{C_i}$$

      where $\mathbf{H}_{C_i} \leftarrow \mathsf{SEvalFX}(\mathsf{pp}, \mathbf{B}^{(i-1)}, C_i, x^{(i)})$.
    * Apply rekey

$$\mathbf{B}^{(i)} = \mathbf{B}^{(i-1)}_{C_i} + [\mathbf{A} \mid \mathbf{Z}] \cdot \mathsf{rk}^{(i)} \quad , \quad \mathbf{C}^{(i)} = \mathbf{C}^{(i-1)}_{C_i} + [\mathbf{C_A} \mid \mathbf{C_Z}] \cdot \mathsf{rk}^{(i)}.$$

  - Compute $\mathbf{C}_{\mathsf{out}} = \mathbf{C}^{(d)} \cdot \mathbf{H}^{(d)}_{\mathsf{out}}$ where $\mathbf{H}^{(d)}_{\mathsf{out}} = \mathsf{RXRead}(\mathsf{pp}, \mathbf{B}^{(d)}, |C|, x^{(d)})$.
  - Return message as $\lfloor \mathbf{C}_\mu - [\mathbf{C_A} \mid \mathbf{C_Z} \mid \mathbf{C}_{\mathsf{out}}] \cdot \mathbf{k}_C \rceil_2$.

9.3 Key Succinctness and Generation Depth

From the succinctness of Definition 7.9 RAM encodings, we can directly observe that the dimensions of our secret keys along with our modulus depend only on the depth of the circuit C on not its size.

The analysis of KeyGen depth follows from an essentially identical argument to that of Construction 7.12, as they utilize the same underlying lattice ingredients and basic structure.

9.4 Correctness

Theorem 9.3 (Correctness)**.** *Construction 9.2 is a correct attribute-based encryption scheme.*

Proof. The argument follows analogous to Section 7.5, where we inductively bound the distance $||\mathbf{C}^{(i)} - \mathbf{s}^\intercal \cdot (\mathbf{B}^{(i)} - h^{(i)} \otimes \mathbf{G})||$, which can in turn be used to bound the difference

$$\left(\mathbf{C}_\mu - \mu \cdot \left\lceil \frac{q}{2} \right\rfloor\right) - ([\mathbf{C_A} \mid \mathbf{C_Z} \mid \mathbf{C}_{\mathsf{out}}] \cdot \mathbf{k}_C)$$

Since decryption simply rounds $\mathbf{C}_\mu - [\mathbf{C_A} \mid \mathbf{C_Z} \mid \mathbf{C}_{\mathsf{out}}] \cdot \mathbf{k}_C$ to $\mathbb{Z}_2$, it suffices to bound the above error to be $< (\frac{q}{2})$.

In this case, we will inductively argue that for $i \in [d]$

$$||\mathbf{C}^{(i)} - \mathbf{s}^\intercal \cdot (\mathbf{B}^{(i)} - h^{(i)} \otimes \mathbf{G})|| \leq \sigma \cdot B^{i+2}$$

Our base case, for $\mathbf{C}^{(0)} = \mathbf{C}_x$, follows from construction, as $\mathbf{C}_x$ is computed as

$$\mathbf{s}^\intercal (\mathbf{B} - h^{(0)} \otimes \mathbf{G}) + \mathbf{e}_0^\intercal \cdot \mathbf{R_B}.$$

Since $||\mathbf{e}_0|| \leq \sigma$ and $||\mathbf{R_B}|| \leq 1$, we can bound the total difference

$$||\mathbf{C}^{(0)} - \mathbf{s}^\intercal \cdot (\mathbf{B}^{(0)} - h^{(0)} \otimes \mathbf{G})|| \leq \sigma \cdot \mathsf{poly}(\lambda) \leq \sigma \cdot B$$

Proceeding onto our inductive step, setting

$$\mathbf{C}^{(i-1)} = \mathbf{s}^\intercal \cdot (\mathbf{B}^{(i-1)} - h^{(i-1)} \otimes \mathbf{G}) + \mathbf{e}^\intercal_{(i-1)}$$

for some bounded error $||\mathbf{e}_{(i-1)}|| \leq \sigma \cdot B^{i+1}$, we can write

$$\begin{aligned}\mathbf{C}_{C_i}^{(i-1)} = [\mathbf{C}^{(i-1)} \mid \mathbf{C}_\mathbf{Z}] \cdot \mathbf{H}_{C_i} &= \mathbf{s}^\top[(\mathbf{B}^{(i-1)} - h^{(i-1)} \otimes \mathbf{G}) \mid \mathbf{Z}] \cdot \mathbf{H}_{C_i} + [\mathbf{e}_{(i-1)}^\top \mid \mathbf{e}_0^\top] \cdot \mathbf{H}_{C_i} \\ &= \mathbf{s}^\top \cdot (\mathbf{B}_{C_i}^{(i-1)} - h^{(i)} \otimes \mathbf{G}) + \mathbf{e}_{(i-1,C_i)}^\top\end{aligned}$$

where $\mathbf{e}_{(i-1,C_i)}^\top = [\mathbf{e}_{(i-1)}^\top \mid \mathbf{e}_0^\top] \cdot \mathbf{H}_{C_i}$, which then gives

$$\begin{aligned}\mathbf{C}^{(i)} &= \mathbf{C}_{C_i}^{(i-1)} + [\mathbf{C}_\mathbf{A} \mid \mathbf{C}_\mathbf{Z}] \cdot \mathsf{rk}^{(i)} \\ &= \mathbf{s}^\top \cdot (\mathbf{B}_{C_i}^{(i-1)} - h^{(i)} \otimes \mathbf{G}) + \mathbf{e}_{(i-1,C_i)}^\top + (\mathbf{s}^\top \cdot [\mathbf{A} \mid \mathbf{Z}] + [\mathbf{e}_1^\top \mid \mathbf{e}_0^\top]) \cdot \mathsf{rk}^{(i)} \\ &= \mathbf{s}^\top \cdot \left(\mathbf{B}_{C_i}^{(i-1)} - h^{(i)} \otimes \mathbf{G} + \left(\mathbf{B}^{(i)} - \mathbf{B}_{C_i}^{(i-1)}\right)\right) + \left(\mathbf{e}_{(i-1,C_i)}^\top + [\mathbf{e}_1^\top \mid \mathbf{e}_0^\top] \cdot \mathsf{rk}^{(i)}\right)\end{aligned}$$

Observe that we can bound $||\mathbf{H}_{C_i}|| \leq \mathsf{poly}(\lambda)$, $||\mathbf{e}_0||, ||\mathbf{e}_1|| \leq \sigma \cdot \mathsf{poly}(\lambda)$, and $||\mathsf{rk}^{(i)}|| \leq \sigma_i \cdot B$. Combining this with the inductive hypothesis gives

$$\left\|\left(\mathbf{e}_{(i-1,C_i)}^\top + [\mathbf{e}_1^\top \mid \mathbf{e}_0^\top] \cdot \mathsf{rk}^{(i)}\right)\right\| \leq (\sigma_i \cdot B + \cdot\sigma \cdot B^{i+1}) \cdot \mathsf{poly}(\lambda) \leq (\sigma \cdot B^{i+2}).$$

Which is sufficient to complete our induction, which lets us bound

$$||\mathbf{C}^{(i)} - \mathbf{s}^\top \cdot (\mathbf{B}^{(i)} - h^{(i)} \otimes \mathbf{G})|| \leq \sigma \cdot B^{d+2}$$

Now, to bound the final error on $[\mathbf{C}_\mathbf{A} \mid \mathbf{C}_\mathbf{Z} \mid \mathbf{C}_{\mathsf{out}}] \cdot \mathbf{k}_C$, it remains is to apply the read homomorphism on the output wire $\mathbf{H}_{\mathsf{out}}^{(d)}$ and the decryption component $\mathbf{k}_\mathbf{C}$, which follow from similar arguments. The error growth here is dominated by $||\mathbf{k}_\mathbf{C}|| \leq \sigma_{\mathsf{out}} = B^{d+1}$, giving us a final error of $\sigma \cdot B^{d+2} \cdot B^{d+1} \cdot \mathsf{poly}(\lambda)$. Since we chose our modulus to superpolynomially larger $\geq \sigma \cdot B^{2d+2}$, correctness holds with overwhelming probability. □

9.5 Security

Theorem 9.4 (Selective Security)**.** *Assuming decomposed LWE, Construction 9.2 satisfies selective security as an attribute-based encryption scheme.*

Proof. Consider the following series of hybrids:

- $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$: This is the real (selective) security game instantiated with Construction 9.2.

  1. **Setup Phase:** The adversary $\mathcal{A}$ specifies an input x^* and bound on circuit depth. The challenger samples the following:
     (a) Sample RAM encoding setup $(\mathsf{pp}, \mathbf{Z}) \leftarrow \mathsf{RSetup}(1^\lambda)$.
     (b) Sample noise vectors $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ as:

$$\mathbf{e}_0 \xleftarrow{\mathrm{R}} \mathcal{D}_\sigma^t \quad , \quad \mathbf{R}_\mathbf{A} \xleftarrow{\mathrm{R}} \{0,1\}^{t \times m} \quad , \quad \mathbf{R}_\mathbf{B} \xleftarrow{\mathrm{R}} \{0,1\}^{m+t \times m \cdot k} \quad , \quad \mathbf{R}_\mathbf{p} \xleftarrow{\mathrm{R}} \{0,1\}^{t \times 1}$$

$$\mathbf{e}_1^\top = \mathbf{e}_0^\top \cdot \mathbf{R}_\mathbf{A} \quad , \quad \mathbf{e}_2^\top = [\mathbf{e}_1 \mid \mathbf{e}_0^\top] \cdot \mathbf{R}_\mathbf{B} \quad , \quad \mathbf{e}_3^\top = \mathbf{e}_0^\top \cdot \mathbf{R}_\mathbf{p}.$$

     (c) Sample trapdoored matrix $(\mathbf{A}, \mathsf{td}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
     (d) Sample attribute encoding matrix $\mathbf{B}^{(0)} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times k \cdot m}$ and message encoding vector $\mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$.
     (e) Set ciphertext components

$$\mathbf{C}_\mathbf{Z} = \mathbf{s}^\top \mathbf{Z} + \mathbf{e}_0^\top \quad , \quad \mathbf{C}_\mathbf{A} = \mathbf{s}^\top \mathbf{A} + \mathbf{e}_1^\top$$

$$\mathbf{C}_x = \mathbf{s}^\top (\mathbf{B} - \mathsf{RHash}(x) \otimes \mathbf{G}) + \mathbf{e}_2^\top \quad , \quad \mathbf{C}_\mu = \mathbf{s}^\top \mathbf{p} + \mathbf{e}_3^\top + b \cdot \left\lceil \frac{q}{2} \right\rfloor.$$

     (f) The challenger returns $(\mathsf{pp}, \mathbf{Z}, \mathbf{A}, \mathbf{B}^{(0)}, \mathbf{p})$ to the adversary.

2. **Keygen Phase:** As many times as desired, adversary $\mathcal{A}$ submits a circuit C of depth d where $C(x^*) = 1$. The challenger responds with as follows:
   (a) For $i \in [d]$, sample base matrices $\mathbf{B}^{(i)} \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$
   (b) For $i \in [d]$, set $\mathbf{B}_{C_i}^{(i-1)} \leftarrow \mathsf{SEvalF}(\mathsf{pp}, \mathbf{B}^{(i-1)}, C_i)$.
   (c) For $i \in [d]$, sample $\mathsf{rk}^{(i)} \leftarrow \mathsf{SamplePre}\left([\mathbf{A} \mid \mathbf{Z}], \begin{bmatrix} \mathsf{td}_{\mathbf{A}} \\ \mathbf{0} \end{bmatrix}, \mathbf{B}_{C_i}^{(i-1)} - \mathbf{B}^{(i)}, \sigma_i\right)$.
   (d) Set $\mathbf{B}_{\mathsf{out}}^{(d)} \leftarrow \mathsf{RRead}(\mathsf{pp}, \mathbf{B}^{(d)}, |C|)$.
   (e) Sample decryption component $\mathbf{k}_C \leftarrow \mathsf{SamplePre}\left([\mathbf{A} \mid \mathbf{Z} \mid \mathbf{B}_{\mathsf{out}}^{(d)}], \begin{bmatrix} \mathsf{td}_{\mathbf{A}} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \mathbf{p}, \sigma_{\mathsf{out}}\right)$.
   (f) The challenger returns $\mathsf{sk}_C = (\{\mathsf{rk}^{(i)}\}_{i \in [d]}, \mathbf{k}_C, \mathsf{mpk})$
3. **Challenge Phase:**[19] The challenger sends ciphertext $(x^*, \mathbf{C}_{\mathbf{Z}}, \mathbf{C}_{\mathbf{A}}, \mathbf{C}_x, \mathbf{C}_\mu)$.
4. **(Post-Challenge) Keygen Phase:** Same as pre-challenge
5. **Output Phase:** Adversary $\mathcal{A}$ submits a bit b', which is the output of the experiment.

- $\mathsf{Hyb}_1^{(b)}$: In this experiment, the challenger sets

  1.(d) Sample attribute encoding matrix $\mathbf{B}^{(0)} = [\mathbf{A} \mid \mathbf{Z}] \cdot \mathbf{R}_{\mathbf{B}} + \mathsf{RHash}(x^*) \otimes \mathbf{G}$ and message encoding vector $\mathbf{p} \xleftarrow{R} \mathbb{Z}_q^n$.

- $\mathsf{Hyb}_{2,i^*}^{(b)}$: For $i^* \in [d]$, in this experiment, for all keygen queries, the challenger samples

  2.(a) For $i \in [d]$ where $i \leq i^*$ sample $\mathbf{R}^{(i)} \xleftarrow{R} \mathcal{D}_{\sigma_i \cdot B}^{(m+t) \times m}$ and set $\mathbf{B}^{(i)} = [\mathbf{A} \mid \mathbf{Z}] \cdot \mathbf{R}^{(i)} + h^{(i)} \otimes \mathbf{G}$. Otherwise, sample $.\mathbf{B}^{(i)} \xleftarrow{R} \mathbb{Z}_q^{n \times k \cdot m}$.

  2.(c) For $i \in [d]$ where $i \leq i^*$ set $\mathsf{rk}^{(i)} = \mathbf{R}^{(i-1)} \cdot \mathbf{H}_{C_i} + \mathbf{R}^{(i)}$. Otherwise, sample

  $$\mathsf{rk}^{(i)} \leftarrow \mathsf{SamplePre}\left([\mathbf{A} \mid \mathbf{Z}], \begin{bmatrix} \mathsf{td}_{\mathbf{A}} \\ \mathbf{0} \end{bmatrix}, \mathbf{B}_{C_i}^{(i-1)} - \mathbf{B}^{(i)}, \sigma_i\right).$$

  where $\mathbf{R}^{(0)} = \mathbf{R}_{\mathbf{B}}$.

- $\mathsf{Hyb}_3^{(b)}$: In this experiment, for all keygen queries, the challenger samples

  2.(e) Sample decryption component

  $$\mathbf{k}_C \leftarrow \mathsf{SamplePre}\left(\left[\mathbf{A} \mid \mathbf{Z} \mid \mathbf{B}_{\mathsf{out}}^{(d)}\right], \begin{bmatrix} -\mathbf{R}_{\mathsf{out}}^{(d)} \\ \mathbf{I}_m \end{bmatrix}, \mathbf{p}, \sigma_{\mathsf{out}}\right)$$

  where

  $$\mathbf{R}_{\mathsf{out}}^{(d)} = \left[\begin{array}{c|c} \mathbf{R}^{(d)} & \begin{bmatrix} \mathbf{0} \\ \mathbf{I}_t \end{bmatrix} \end{array}\right] \cdot \mathbf{H}_{\mathsf{out}}^{(d)}$$

  (recall from our decryption relation $\mathbf{H}_{\mathsf{out}}^{(d)} \leftarrow \mathsf{RXRead}(\mathsf{pp}, \mathbf{B}^{(d)}, |C|, x^{(d)})$).

- $\mathsf{Hyb}_4^{(b)}$: In this experiment, the challenger samples $\mathbf{A}$ uniformly.

  1.(c) Sample ~~trapdoored~~ matrix $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$.

- $\mathsf{Hyb}_5^{(b)}$: In this experiment, the challenger computes $\mathbf{A}, \mathbf{p}$ from $\mathbf{Z}$.

[19] Since we present our scheme as a bit-encryption, the adversary is simply tasked with distinguishing encryptions of 0 from 1

1.(c) Sample matrix $\mathbf{A} = \mathbf{Z} \cdot \mathbf{R}_{\mathbf{A}}$.

1.(d) Sample attribute encoding matrix $\mathbf{B}^{(0)} = [\mathbf{A} \mid \mathbf{Z}] \cdot \mathbf{R}_{\mathbf{B}} + \mathsf{RHash}(x^*) \otimes \mathbf{G}$ and message encoding vector $\mathbf{p} = \mathbf{Z} \cdot \mathbf{R}_{\mathbf{p}}$.

- $\mathsf{Hyb}_6^{(b)}$: In this experiment, the challenger computes $\mathbf{C}_{\mathbf{A}}, \mathbf{C}_x, \mathbf{C}_\mu$ from $\mathbf{C}_{\mathbf{Z}}$.

  1.(e) Set ciphertext components

$$\mathbf{C}_{\mathbf{Z}} = \mathbf{s}^\top \mathbf{Z} + \mathbf{e}_0^\top \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{A}}$$

$$\mathbf{C}_x = [\mathbf{C}_{\mathbf{A}} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{R}_{\mathbf{B}} + \mathbf{e}_2^\top \quad , \quad \mathbf{C}_\mu = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{p}} + \mu \cdot \left\lceil \frac{q}{2} \right\rfloor .$$

- $\mathsf{Hyb}_7^{(b)}$: In this experiment, the challenger samples $\mathbf{C}_{\mathbf{Z}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^t$.

  1.(e) Set ciphertext components

$$\mathbf{C}_{\mathbf{Z}} = \xleftarrow{\mathrm{R}} \mathbb{Z}_q^t \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{A}}$$

$$\mathbf{C}_x = [\mathbf{C}_{\mathbf{A}} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{R}_{\mathbf{B}} + \mathbf{e}_2^\top \quad , \quad \mathbf{C}_\mu = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{p}} + \mu \cdot \left\lceil \frac{q}{2} \right\rfloor .$$

- $\mathsf{Hyb}_8^{(b)}$: In this experiment, the challenger samples $\mathbf{C}_\mu \xleftarrow{\mathrm{R}} \mathbb{Z}_q$.

  1.(e) Set ciphertext components

$$\mathbf{C}_{\mathbf{Z}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^t \quad , \quad \mathbf{C}_{\mathbf{A}} = \mathbf{C}_{\mathbf{Z}} \cdot \mathbf{R}_{\mathbf{A}}$$

$$\mathbf{C}_x = [\mathbf{C}_{\mathbf{A}} \mid \mathbf{C}_{\mathbf{Z}}] \cdot \mathbf{R}_{\mathbf{B}} + \mathbf{e}_2^\top \quad , \quad \mathbf{C}_\mu \xleftarrow{\mathrm{R}} \mathbb{Z}_q.$$

Lemma 9.5. *For all adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Hyb}_{\mathsf{real}}^{(b)}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}, \lambda) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a straightforward application of the leftover hash lemma. □

Lemma 9.6. *For all adversaries $\mathcal{A}$,*

$$\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}, \lambda) = 1] = \Pr[\mathsf{Hyb}_{2,0}^{(b)}(\mathcal{A}, \lambda) = 1].$$

Proof. These hybrids are identical. □

Lemma 9.7. *For all adversaries $\mathcal{A}$ and $i \in [d]$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Hyb}_{2,i-1}^{(b)}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(b)}(\mathcal{A}, \lambda) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. We can further break this hybrid transition down into a series of subhybrids:

- sHyb_1: In this experiment, we explicitly sample the difference $(\mathbf{B}^{(i)} - \mathbf{B}_{C_i}^{(i-1)})$ as $\mathbf{T}^{(i)} \leftarrow \mathbb{Z}_q^{n \times k \cdot m}$ and set $\mathbf{B}^{(i)} = \mathbf{T}^{(i)} + \mathbf{B}_{C_i}^{(i-1)}$. Since $\mathbf{B}^{(i)}$ is sampled uniformly in $\mathsf{Hyb}_{2,i-1}$, these distributions are identical.

- sHyb_2: In this experiment, we sample $\mathsf{rk}^{(i)} \xleftarrow{\mathrm{R}} \mathcal{D}_{B \cdot \sigma_i}^{(m+t) \times (k \cdot m)}$ and set $\mathbf{T}^{(i)} = [\mathbf{A} \mid \mathbf{Z}] \cdot \mathsf{rk}^{(i)}$. This follows from the preimage-distribution property of the gadget trapdoor.

- sHyb_3: In this experiment, we sample $\mathbf{R}^{(i)} \xleftarrow{\mathrm{R}} \mathcal{D}_{B \cdot \sigma_i}^{(m+t) \times (k \cdot m)}$ and set $\mathsf{rk}^{(i)} = \mathbf{R}^{(i-1)} \cdot \mathbf{H}_{C_i} + \mathbf{R}^{(i)}$. Since $B \cdot \sigma_i \gg ||\mathbf{R}^{(i-1)} \cdot \mathbf{H}_{C_i}|| = B \cdot \sigma_{i-1} \cdot \mathsf{poly}(\lambda)$, this follows by noise smudging (recall as a base case $\sigma_0 = 1$, corresponding to the binary $\mathbf{R}^{(0)} = \mathbf{R}_{\mathbf{B}}$).

Since each KeyGen query samples its own set of $\mathbf{B}^{(i)}, \mathsf{rk}^{(i)}$, we iteratively apply this series of subhybrids to the output of every KeyGen query. □

Lemma 9.8. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}^{(b)}_{2,d}(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Hyb}^{(b)}_{3}(\mathcal{A},\lambda)=1]\right|=\mathsf{negl}(\lambda).$$

Proof. Recall by $\mathsf{Hyb}_{2,d}$, we compute

$$\mathbf{B}^{(d)}=[\mathbf{A}\mid\mathbf{Z}]\cdot\mathbf{R}^{(d)}+h^{(d)}\otimes\mathbf{G}$$

Thus, by the homomorphism of the RAM encoding we have

$$\begin{aligned}
&[\mathbf{B}^{(d)}-h^{(d)}\otimes\mathbf{G}\mid\mathbf{Z}]\cdot\mathbf{H}_{\mathsf{out}}=\mathbf{B}^{(d)}_{\mathsf{out}}-1\cdot\mathbf{G}\\
\Leftrightarrow\ &[[\mathbf{A}\mid\mathbf{Z}]\cdot\mathbf{R}^{(d)}\mid\mathbf{Z}]\cdot\mathbf{H}_{\mathsf{out}}=\mathbf{B}^{(d)}_{\mathsf{out}}-\mathbf{G}\\
\Leftrightarrow\ &[\mathbf{A}\mid\mathbf{Z}]\cdot\left[\mathbf{R}^{(d)}\;\middle|\;\begin{bmatrix}\mathbf{0}\\ \mathbf{I}_t\end{bmatrix}\right]\cdot\mathbf{H}_{\mathsf{out}}-\mathbf{B}^{(d)}_{\mathsf{out}}=-\mathbf{G}\\
&[\mathbf{A}\mid\mathbf{Z}\mid\mathbf{B}^{(d)}_{\mathsf{out}}]\cdot\begin{bmatrix}-\mathbf{R}^{(d)}_{\mathsf{out}}\\ \mathbf{I}_m\end{bmatrix}=\mathbf{G}
\end{aligned}$$

Since $\begin{bmatrix}-\mathbf{R}^{(d)}_{\mathsf{out}}\\ \mathbf{I}_m\end{bmatrix}$ is indeed a gadget trapdoor with entries bounded by $||\mathbf{R}^{(d)}||\cdot||\mathbf{H}_{\mathsf{out}}||\cdot\mathsf{poly}(\lambda)\ll||\mathbf{R}^{(d)}||\cdot B=\sigma_{\mathsf{out}}$, the distribution of $\mathbf{k}_C$ is both $\mathsf{Hyb}_{2,d}$, Hyb_3 are both uniform preimages drawn from $\mathcal{D}_{\sigma_{\mathsf{out}}\cdot B}$ of a random $\mathbf{p}$ and thus identically distributed. As with Lemma 9.7, we iteratively apply this argument to the $\mathbf{k}_C$ output by every KeyGen query. □

Lemma 9.9. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}^{(b)}_{3}(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Hyb}^{(b)}_{4}(\mathcal{A},\lambda)=1]\right|=\mathsf{negl}(\lambda).$$

Proof. Now that $\mathsf{td}_{\mathbf{A}}$ is no longer being used, so by trapgen uniformity the distribution of $\mathbf{A}$ is uniform. □

Lemma 9.10. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}^{(b)}_{4}(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Hyb}^{(b)}_{5}(\mathcal{A},\lambda)=1]\right|=\mathsf{negl}(\lambda).$$

Proof. This follows from a straightforward application of the leftover hash lemma (via the LHL compatibility of Definition 7.9). □

Lemma 9.11. *For all adversaries $\mathcal{A}$,*

$$\Pr[\mathsf{Hyb}^{(b)}_{5}(\mathcal{A},\lambda)=1]=\Pr[\mathsf{Hyb}^{(b)}_{6}(\mathcal{A},\lambda)=1].$$

Proof. Observe the differences between these hybrids are entirely syntactic, and can be rewritten (See Lemma 7.20). □

Lemma 9.12. *Assuming decomposed LWE, for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}^{(b)}_{6}(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Hyb}^{(b)}_{7}(\mathcal{A},\lambda)=1]\right|=\mathsf{negl}(\lambda).$$

Proof. Specifically, this will follow from the LWE indistinguishability property of the key-homomorphic RAM encoding, where

$$\{\mathsf{pp},\mathbf{Z},\mathbf{s}^{\intercal}\cdot\mathbf{Z}+\mathbf{e}^{\intercal}\}\approx\{\mathsf{pp},\mathbf{Z},\mathbf{t}^{\intercal}\}.$$

Observe at by $\mathsf{Hyb}^{(b)}_6$, all the elements in the view of the adversary dependent on $\mathbf{s}$ can be derived from $\mathbf{C}_{\mathbf{Z}}=\mathbf{s}^{\intercal}\cdot\mathbf{Z}+\mathbf{e}^{\intercal}_0$. This can be directly replaced with the (decomposed) LWE challenge vector. □

Lemma 9.13. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}_7^{(b)}(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Hyb}_8^{(b)}(\mathcal{A},\lambda)=1]\right|=\mathsf{negl}(\lambda).$$

Proof. Since $\mathbf{C}_\mathbf{Z}$ is now a uniformly random vector,

$$(\mathbf{C}_\mathbf{Z}\cdot\mathbf{R}_\mathbf{p},\mathbf{e}_0^\intercal\cdot\mathbf{R}_\mathbf{p})\approx_s(r\xleftarrow{R}\mathbb{Z}_q^n,\mathbf{e}_0^\intercal\cdot\mathbf{R}_\mathbf{p})$$

follows from the leftover hash lemma. □

Lemma 9.14. *For all adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathsf{Hyb}_8^{(0)}(\mathcal{A},\lambda)=1]=\Pr[\mathsf{Hyb}_8^{(1)}(\mathcal{A},\lambda)=1].$$

Proof. Since $\mathbf{C}_\mathbf{p}$ is uniformly random, the adversary's view is now independent of the challenge bit b. □

Taking together Lemmas 9.5 to 9.14 gives us the proof of Theorem 9.4 □

10 Acknowledgments

Huijia Lin is supported by a Simons collaboration grant for algorithmic fairness and Amazon Research Award. We thank Lawrence Roy for pointing out an error the group-based garbling schemes in an earlier version of this work.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.

[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, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *computational complexity*, 15(2):115–162, 2006.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP 2010, Part I*, volume 6198 of *LNCS*, pages 152–163, Bordeaux, France, July 6–10, 2010. Springer Berlin Heidelberg, Germany.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108, Philadephia, PA, USA, May 22–24, 1996. ACM Press.

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Key-homomorphic computations for ram: Fully succinct randomised encodings and more. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, 2025.

[App17] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 1–44. Springer, 2017.

[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 Journal on Computing*, 47(3):1123–1210, 2018.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[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, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany.

[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, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[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, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[BLLL23] Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 3–34, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[CDL01] Andrew Chiu, George Davida, and Bruce Litow. Division in logspace-uniform nc1. *RAIRO-Theoretical Informatics and Applications*, 35(3):259–275, 2001.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer Berlin Heidelberg, Germany.

[DKK18] Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 213–242, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany.

[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, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[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, Denver, CO, USA, October 12–16, 2015. ACM Press.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. In *Elect. Colloq. on Computational Complexity (ECCC)*, volume 7, 2000.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In Chris Umans, editor, *58th FOCS*, pages 588–599, Berkeley, CA, USA, October 15–17, 2017. IEEE Computer Society Press.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[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, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany.

[HK21] David Heath and Vladimir Kolesnikov. One hot garbling. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 574–593, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[HKO22] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. EpiGRAM: Practical garbled RAM. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 3–33, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[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, Santa Cruz, CA, USA, November 6–9, 2023. IEEE Computer Society Press.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP 2002*, volume 2380 of *LNCS*, pages 244–256, Malaga, Spain, July 8–13, 2002. Springer Berlin Heidelberg, Germany.

[ILL25a] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct homomorphic macs from groups and applications. In Ran Raz and Rotem Oshman, editors, *In 66th FOCS*, 2025.

[ILL25b] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, 2025.

[ILL25c] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. Cryptology ePrint Archive, Report 2025/442, 2025.

[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, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662, Copenhagen, Denmark, July 8–11, 2014. Springer Berlin Heidelberg, Germany.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966, Berlin, Germany, November 4–8, 2013. ACM Press.

[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, Portland, OR, USA, June 14–17, 2015. ACM Press.

[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, Santa Barbara, CA, USA, August 17–21, 2014. Springer Berlin Heidelberg, Germany.

[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, Reykjavik, Iceland, July 7–11, 2008. Springer Berlin Heidelberg, Germany.

[Lin25] Robin Linus. Bitvm 3s – garbled circuits for efficient computation on bitcoin. Technical report, BitVM Alliance, 2025.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.

[LWYY25] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: Garbled circuits with 1 bit per gate. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 437–466, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 71–97, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland.

[MORS25a] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Computationally succinct authentication from dcr: Attribute-based laconic function evaluation and more. Manuscript, 2025.

[MORS25b] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from dcr. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, 2025.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[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, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.

[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, Tokyo, Japan, December 6–10, 2009. Springer Berlin Heidelberg, Germany.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870, Paris, France, October 7–9, 2018. IEEE Computer Society Press.

[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 16–20, 2021. Springer, Cham, Switzerland.

[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 16–20, 2021. Springer, Cham, Switzerland.

[Wee24] Hoeteck Wee. Circuit ABE with poly(depth, λ)-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 178–209, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[Wee25] Hoeteck Wee. Almost optimal KP and CP-ABE for circuits from succinct LWE. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 34–62, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 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, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.