


# Authenticated BitGC for Actively Secure Rate-One 2PC

Hanlin Liu 

Northwestern University

[hanlin.liu@northwestern.edu](mailto:hanlin.liu@northwestern.edu)

Xiao Wang 

Northwestern University

[wangxiao@northwestern.edu](mailto:wangxiao@northwestern.edu)

Kang Yang 

State Key Laboratory of Cryptology

[yangk@sklc.org](mailto:yangk@sklc.org)

Yu Yu 

Shanghai Jiao Tong University

Shanghai Qi Zhi Institute

[yuyu@yuyu.hk](mailto:yuyu@yuyu.hk)

## Abstract

In this paper, we present a constant-round actively secure two-party computation protocol with small communication based on the ring learning with errors (RLWE) assumption with key-dependent message security. Our result builds on the recent BitGC protocol by Liu, Wang, Yang, and Yu (Eurocrypt 2025) with communication of one bit per gate for semi-honest security. First, we achieve a different manner of distributed garbling, where the global correlation is secret-shared among the two parties. The garbler always and only holds the garbled labels corresponding to the wire values when all inputs are zero, while the evaluator holds the labels corresponding to the real evaluation. In the second phase, we run an authentication protocol that requires some extra communication, which allows two parties to check the correct computation of each gate by treating the ciphertext as commitments, now that the global key is distributed. For layered circuits, the extra communication for authentication is  $o(1)$  bits per gate, resulting in total communication of  $1 + o(1)$  bits per gate. For generic circuits, the extra communication cost can be 1 bit per gate in the worst case, and thus, the total communication cost would be 2 bits per gate.

## 1 Introduction

Secure two-party computation (2PC) with active security is an important line of works with both theoretical importance and, more recently, practical applications [ZMM<sup>+</sup>20, CDH<sup>+</sup>23, BBC<sup>+</sup>24]. The existence of actively secure 2PC is known since the original garbled-circuit protocol [Yao86] and the GMW compiler [GMW87], whose concrete efficiency was recently explored [ASH<sup>+</sup>20]. The first provably secure and concretely efficient protocol was proposed by Lindell and Pinkas [LP07] using the “cut-and-choose” technique. Its concrete efficiency was later improved by a sequence of works such as [sS11, sS13, KsS12, LP11, HKE13, Lin13, Bra13, AMPR14, WMK17], and the latest protocol requires exactly  $\rho$  garbled circuits to achieve  $\rho$ -bit statistical security. This means  $1.5\rho\lambda$  bits per gate with the recent garbling scheme [RR21] for  $\lambda$ -bit computational security. Another line of work, LEGO [NO09, FJN<sup>+</sup>13, FJNT15, NST17, KNR<sup>+</sup>17] improved the asymptotic communication to about  $O(\rho\lambda/\log C)$  bits per gate with comparable concrete efficiency, where  $C$  is the circuit size.

On the other hand, smaller communication can be achieved when the communication rounds are linear to the depth of the circuit. TinyOT [NNOB12] was among the first to have concretely efficient and actively secure protocols for GMW [GMW87], which was later optimized by a series of works (e.g., [FKOS15, BLN<sup>+</sup>21]). The best-known approach to generate TinyOT-like preprocessing correlations is to use pseudorandom correlation generators [BCG<sup>+</sup>19, LXYY25] with communication cost sublinear to the number of correlations. As a result, TinyOT requires communication of  $4 + o(1)$  bits per gate. Another non-constant-round solution

is the IPS compiler [IPS08], which requires communication of  $O(1)$  bits per gate, although the underlying constant is conjectured to be fairly large.

The state-of-the-art constant-round actively secure 2PC is the authenticated garbling framework [WRK17a, WRK17b, HSS17, KRRW18, YWZ20, DILO22a, CWYY23] that transmits only one garbled circuit, using preprocessing similar to TinyOT, thus cheap preprocessing and constant-round at the same time. The authenticated-garbling approach needs communication of  $O(\lambda)$  bits per gate. Using heavier mechanisms, actively secure constant-round 2PC could also be constructed with even smaller communication only linear to the input size by using a fully homomorphic encryption (FHE) scheme [Gen09, BGV12] or a laconic function-hiding functional encryption (LFE) scheme [QWW18, HLL23], and a zkSNARK scheme [Gro10] to prove the FHE/LFE computation in a non-black-box way. The only protocol that avoids using zkSNARK while obtaining succinct communication is by Morgan et al. [MPP20] that is also impressively non-interactive; however, it appears to be a feasibility result, not targeting concrete efficiency. In conclusion, there is a huge middle ground between concretely efficient constant-round 2PC with active security, which can compute more than one million gates per second with  $O(\lambda)$  bits per gate, and sublinear communication solutions that deploy zkSNARK on top of FHE/LFE.

## 1.1 Our Contribution

In this paper, we propose the first concretely efficient constant-round 2PC protocol with active security, achieving total communication of  $o(\lambda C)$  bits. To be more concrete, the total communication is  $2C + O(\lambda^2 \log \lambda)$  bits for general circuits, or  $C + o(C) + O(\lambda^2 \log \lambda)$  bits for layered circuits. Our solution builds upon the recent BitGC protocol with passive security by Liu, Wang, Yang, and Yu [LWYY25], which requires communication of only 1 bit per gate. We make the following contributions to enhance the security of BitGC to active security.

1. First, we distribute the BitGC protocol so that no party has the global correlated key between zero and one labels. By doing this, the garbler essentially behaves like an evaluator with all-zero input labels while the evaluator has real input labels.
2. Then, we present a lightweight authentication protocol to check the correctness of the computation, requiring one extra bit of communication per gate. The check reduces to checking if a list of ciphertexts all encrypt zero, which can be performed efficiently.
3. Finally, we show actively secure two-party key-generation and encryption protocols for the lattice-based encryption scheme, namely extended GSW, needed by the above main protocol. These protocols need to be executed only once in the setup phase for life between a pair of two parties.

Our scheme could also be extended to achieve even smaller communication: if the underlying somewhat homomorphic encryption scheme can evaluate  $L$  extra levels of multiplications beyond what is required in the original BitGC protocol, then we can achieve communication of  $1 + O(1/L)$  bits per gate for layered circuits, or communication of  $1 + O(\log C/L)$  bits per gate for arbitrary circuits. When  $L$  is set as  $\omega(1)$  for layered circuits or  $\omega(\log C)$  for generic circuits, the communication cost is compressed to  $1 + o(1)$  bits per gate, without the need of bootstrapping.

## 2 Technical Overview

Let  $\mathcal{R}_p \stackrel{\text{def}}{=} \mathbb{Z}_p[X]/(X^n + 1)$  and  $\mathcal{R}_q \stackrel{\text{def}}{=} \mathbb{Z}_q[X]/(X^n + 1)$ . For any ring element  $A$ , we define  $\text{LSB}(A)$  as  $A[1] \bmod 2$ , where  $A[1]$  is the first coefficient of  $A$ .

## 2.1 BitGC: Garbled Circuits with 1 Bit per Gate

First, we recall some important details of the BitGC garbling scheme. In the BitGC scheme [LWYY25], all garbled labels are no longer bit strings but ring elements in  $\mathcal{R}_p$ ; there is a global offset  $\Delta \in \mathcal{R}_p$  with  $\text{LSB}(\Delta) = 1$ . For a wire with two labels  $A^0$  and  $A^1$ , we always have  $A^1 = A^0 + (-1)^{\pi_a} \cdot \Delta \pmod p$ , where  $\pi_a = \text{LSB}(A^0)$  is the wire mask. With these definitions, the following holds for any  $v \in \{0, 1\}$ :

$$A^v = A^0 + (-1)^{\pi_a} \cdot v \cdot \Delta, \quad \text{LSB}(A^v) = \pi_a \oplus v, \quad A^v = A^{\pi_a} + \text{LSB}(A^v) \cdot \Delta.$$

**Properties of Extended GSW.** To enable small communication, they extended the GSW scheme [GSW13] to allow some special distributed evaluation. For a bit  $m$ , we use  $\llbracket m \rrbracket$  to denote the ciphertext encrypting  $m$  using  $\Delta$  as the key; if not stated otherwise, all ciphertexts are encrypted under  $\Delta$ . From the original GSW scheme, we already have that for any bit  $m$ ,  $\text{Dec}(\Delta, \llbracket m \rrbracket) = m \cdot \Delta$ . We also assume that  $\llbracket m \rrbracket$  can perform some constant number of homomorphic multiplications and a polynomial number of homomorphic additions.

We briefly talk about the extra properties that they added without going into the construction. For any ciphertexts  $\tau_1, \tau_2, \tau_3$  and uniform  $X, Y$  such that  $\text{LSB}(X) = \text{LSB}(Y) = 0$ , the following properties hold with overwhelming probability:

I Dec is an “odd function”, satisfying  $\text{Dec}(X, -\tau_1) = -1 \cdot \text{Dec}(X, \tau_1)$ .

II Dec supports distributed decryption, where

$$\text{Dec}(X + \Delta, \tau_1) = \text{Dec}(X, \tau_1) + \text{Dec}(\Delta, \tau_1).$$

III Eval is also an “odd function”, satisfying

$$\text{Eval}(X, Y, -\tau_1, -\tau_2, -\tau_3) = -1 \cdot \text{Eval}(X, Y, \tau_1, \tau_2, \tau_3).$$

IV Eval supports “distributed evaluation”, where

$$\begin{aligned} & \text{Eval}(X + i \cdot \Delta, Y + j \cdot \Delta, \tau_1, \tau_2, \tau_3) - \text{Eval}(X, Y, \tau_1, \tau_2, \tau_3) \\ &= i \cdot \text{Dec}(\Delta, \tau_1) + j \cdot \text{Dec}(\Delta, \tau_2) + i \cdot j \cdot \text{Dec}(\Delta, \tau_3). \end{aligned}$$

The construction of Eval requires ciphertexts encrypting key-dependent information, so BitGC relies on the assumption of KDM security.

**The BitGC garbled table and its evaluation.** Assume that a gate with input wire index  $(a, b)$ , output wire index  $c$ , and  $g : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$  is the gate evaluation function. The garbled table associated with this gate consists of three ciphertexts.

$$\begin{aligned} \tau_1 &= \llbracket (-1)^{\pi_c} \cdot (z_{1,0} - z_{0,0}) \rrbracket \\ \tau_2 &= \llbracket (-1)^{\pi_c} \cdot (z_{0,1} - z_{0,0}) \rrbracket \\ \tau_3 &= \llbracket (-1)^{\pi_c} \cdot (z_{0,0} + z_{1,1} - z_{1,0} - z_{0,1}) \rrbracket \end{aligned}$$

where  $z_{i,j} = g(\pi_a \oplus i, \pi_b \oplus j)$  is the actual output wire value when the input masked Boolean values are  $(i, j)$ .

For correct evaluation, the evaluator maintains the invariant that it always has the label corresponding to the actual wire value. This means that it always has  $A^{v_a}$  and  $B^{v_b}$  and wants to obtain  $C^{g(v_a, v_b)}$ . Because for

any bits  $i, j$ , we know  $A^i = A^{\pi_a} + (\pi_a \oplus i) \cdot \Delta$  and that  $B^j = B^{\pi_b} + (\pi_b \oplus j) \cdot \Delta$ . Now using [IV](#), we can obtain that

$$\begin{aligned}
& \text{Eval}(A^i, B^j, \tau_1, \tau_2, \tau_3) - \text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3) \\
&= (-1)^{\pi_c} \cdot \Delta \cdot \left( (\pi_a \oplus i) \cdot (z_{1,0} - z_{0,0}) + (\pi_b \oplus j) \cdot (z_{0,1} - z_{0,0}) \right. \\
&\quad \left. + (\pi_a \oplus i) \cdot (\pi_b \oplus j) \cdot (z_{0,0} + z_{1,1} - z_{1,0} - z_{0,1}) \right) \\
&= (-1)^{\pi_c} \cdot (z_{\pi_a \oplus i, \pi_b \oplus j} - z_{0,0}) \cdot \Delta.
\end{aligned} \tag{1}$$

If the garbler sets  $C^{z_{0,0}} = \text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3)$ , then what the evaluator can compute is

$$\begin{aligned}
\text{Eval}(A^{v_a}, B^{v_b}, \tau_1, \tau_2, \tau_3) &= C^{z_{0,0}} + (-1)^{\pi_c} \cdot (z_{\pi_a \oplus v_a, \pi_b \oplus v_b} - z_{0,0}) \cdot \Delta \\
&= C^{z_{\pi_a \oplus v_a, \pi_b \oplus v_b}} = C^{g(v_a, v_b)}.
\end{aligned}$$

This would allow the evaluator to continue the gate evaluation as long as the garbler sets output labels appropriately during the garbling phase.

**Transmitting BitGC garbled table with 1 bit per gate.** To garble a gate with 1-bit communication, the garbler needs to transmit the ciphertexts  $\tau_1, \tau_2, \tau_3$  with only 1 bit. First of all, transmitting encryptions of random bits is “free” in communication since the garbler can send an encrypted seed  $\llbracket s \rrbracket$ , and the evaluator can homomorphically expand it using a PRG to obtain many pseudorandom bit encryptions. It only requires somewhat homomorphism if a low-complexity PRG is used.

The garbler maintains the invariant that for each gate, both parties have **the same ciphertext** for the permutation bits, namely  $\llbracket \pi_a \rrbracket$  and  $\llbracket \pi_b \rrbracket$ , where  $\pi_a = \text{LSB}(A^0)$  and  $\pi_b = \text{LSB}(B^0)$ . Additionally, both parties have a random bit encryption associated with the output wire, namely  $\llbracket r_c \rrbracket$ .

With this setup, the garbler can garble this gate as follows.

1. **Compute unfixed garbled table.** Both parties can perform homomorphic evaluations on  $\llbracket \pi_a \rrbracket, \llbracket \pi_b \rrbracket$  and  $\llbracket r_c \rrbracket$  to obtain:

$$\begin{aligned}
\tilde{\tau}_1 &= \llbracket (-1)^{r_c} \cdot z_{1,0} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket \\
\tilde{\tau}_2 &= \llbracket (-1)^{r_c} \cdot z_{0,1} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket \\
\tilde{\tau}_3 &= \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket + \llbracket (-1)^{r_c} \cdot z_{1,1} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{1,0} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,1} \rrbracket.
\end{aligned} \tag{2}$$

Here,  $\llbracket (-1)^{r_c} \cdot z_{i,j} \rrbracket = \llbracket (1 - 2r_c) \cdot z_{i,j} \rrbracket = (1 - 2 \cdot \llbracket r_c \rrbracket) \cdot g(\llbracket \pi_a \rrbracket \oplus i, \llbracket \pi_b \rrbracket \oplus j)$  for all bits  $i, j$ .  $\tilde{\tau}_1, \tilde{\tau}_2, \tilde{\tau}_3$  are similar to the final table but use the  $(-1)^{r_c}$  term instead of  $(-1)^{\pi_c}$ . This step requires homomorphic evaluation on a circuit of depth 2.

2. **Compute the output wire mask  $\pi_c$ .** Recall that  $C^{z_{0,0}}$  is defined based on  $\tau_1, \tau_2, \tau_3$ ; therefore  $\pi_c$  is also dependent on them:

$$\pi_c = \text{LSB}(C^{z_{0,0}}) \oplus z_{0,0} = \text{LSB}(\text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3)) \oplus z_{0,0}.$$

However,  $\pi_c$  is needed above to compute  $\tau_i$ . Therefore, the key of this step is to compute  $\pi_c$  without  $\tau_i$ 's. The crucial observation made by the authors [\[LWYY25\]](#) here is that when setting the plaintext space to be even, the LSB on both sides would be the same due to [III](#), meaning that  $\pi_c = \text{LSB}(\text{Eval}(A^{\pi_a}, B^{\pi_b}, -1 \cdot \tau_1, -1 \cdot \tau_2, -1 \cdot \tau_3)) \oplus z_{0,0}$ . Thus regardless of the truth value of  $r_c$ ,  $\pi_c$  can always be computed by  $\pi_c = \text{LSB}(\text{Eval}(A^{\pi_a}, B^{\pi_b}, \tilde{\tau}_1, \tilde{\tau}_2, \tilde{\tau}_3)) \oplus z_{0,0}$ , which the garbler can do based on the output of step 1.

3. **Transmit a single bit.** Now that the garbler computes  $\pi_c$ , it can send  $d_c = r_c \oplus \pi_c$  to the evaluator. Then, both parties compute  $\llbracket \pi_c \rrbracket = d_c \oplus \llbracket r_c \rrbracket$  and update  $\tau_i = (-1)^{d_c} \cdot \tilde{\tau}_i$  locally.
4. **Obtain output label  $C^{\pi_c}$ .** The only invariant to be maintained is letting the garbler obtain  $C^{\pi_c}$ . The garbler can first recover the garbled tables  $\tau_i$ 's just as an evaluator would do; then, as described above, set  $C^{z_{0,0}} = \text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3)$ . This means  $C^{\pi_c} = C^{z_{0,0}} - \text{LSB}(C^{z_{0,0}}) \cdot \Delta$ .

Their work also includes an instantiation of the encryption scheme such that it satisfies all properties on Dec and Eval. Since they are not relevant to our actively secure protocol, we refer readers to their paper [LWYY25] for details.

## 2.2 Distributing BitGC

Now, we discuss how we can strengthen their protocol for active security with a small overhead.

**BitGC is free of selective-failure attack.** Intuitively, making BitGC actively secure should be easier than classical GCs as there is less information being transmitted from the garbler, hence fewer ways to cheat. Indeed, our first observation is that BitGC is already free of selective failure attacks. To be more specific, we can show that if a garbler sends a wrong bit for a gate  $g$ , it is equivalent to changing the gate function to be  $\widehat{g}(\cdot, \cdot) = g(\cdot, \cdot) \oplus 1$ .

Let's first recall the dependency of all values. Recall that for a gate with input labels  $(A^0, A^1), (B^0, B^1)$  and ciphertexts  $(\llbracket \pi_a \rrbracket, \llbracket \pi_b \rrbracket, \llbracket r_c \rrbracket)$ , the garbler first computes  $\pi_c$  and sends the difference  $d_c = \pi_c \oplus r_c$  so that both parties have  $\llbracket \pi_c \rrbracket = d_c \oplus \llbracket r_c \rrbracket$ . Then the garbled table  $\tau_1, \tau_2, \tau_3$  can be computed from  $(\llbracket \pi_a \rrbracket, \llbracket \pi_b \rrbracket, \llbracket r_c \rrbracket, d_c)$ . From here, the evaluator obtains  $C^{v_c} = \text{Eval}(A^{v_a}, B^{v_b}, \tau_1, \tau_2, \tau_3)$ , and the garbler obtain  $C^{z_{0,0}} = \text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3)$ , which means that

$$\begin{aligned} C^0 &= \text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3) - (-1)^{\pi_c} \cdot z_{0,0} \cdot \Delta \\ C^1 &= \text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3) + (-1)^{\pi_c} \cdot \overline{z_{0,0}} \cdot \Delta. \end{aligned}$$

Let's see the effect if the garbler sends  $\widehat{d}_c = \overline{d_c}$ , which is the only way that a garbler can cheat. Let's take a look at the first gate where the garbler cheats. In this case, both parties would obtain  $\llbracket \overline{\pi_c} \rrbracket$ , and thus effectively the new wire mask is  $\widehat{\pi}_c = \overline{\pi_c}$  and the resulting garbled rows would be  $\widehat{\tau}_i = -1 \cdot \tau_i$ .

Now from the evaluator's view, the label that the evaluator would obtain is

$$\begin{aligned} \widehat{C}^{v_c} &= \text{Eval}(A^{v_a}, B^{v_b}, \widehat{\tau}_1, \widehat{\tau}_2, \widehat{\tau}_3) = -\text{Eval}(A^{v_a}, B^{v_b}, \tau_1, \tau_2, \tau_3) \\ &= -C^{v_c}. \end{aligned}$$

From the garbler's perspective, the output labels are:

$$\begin{aligned} \widehat{C}^0 &= -\text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3) - (-1)^{\overline{\pi_c}} \cdot z_{0,0} \cdot \Delta = -C^0 \\ \widehat{C}^1 &= -\text{Eval}(A^{\pi_a}, B^{\pi_b}, \tau_1, \tau_2, \tau_3) + (-1)^{\overline{\pi_c}} \cdot \overline{z_{0,0}} \cdot \Delta = -C^1. \end{aligned}$$

This means that the garbler would obtain two new labels that are  $(-1)$  times the original labels. With the new wire mask  $\widehat{\pi}_c = \overline{\pi_c}$ ,  $\widehat{C}^1$  is new 0-label, because  $\text{LSB}(\widehat{C}^1) = \text{LSB}(-C^1) = \text{LSB}(C^1) = 1 - \text{LSB}(C^0) = \overline{\pi_c}$ . Furthermore, recall that  $C^{v_c} = C^1 - (-1)^{\pi_c} \cdot \overline{v_c} \cdot \Delta$ ; thus we have

$$\widehat{C}^{v_c} = -C^1 + (-1)^{\pi_c} \cdot \overline{v_c} \cdot \Delta = \widehat{C}^1 + (-1)^{\overline{\pi_c}} \cdot \overline{v_c} \cdot \Delta,$$

which means that under the new definition, the evaluator's label is now effectively for the bit  $\overline{v_c}$ , not  $v_c$ .

Effectively, when the garbler sends a flipped bit for a gate, with the evaluator performing the same computation, the garbler would have computed locally two new labels such that all invariants still hold but the permutation bit is flipped and that the real wire value is also flipped. Note that the masked bit is unchanged since  $\text{LSB}(\widetilde{C^{v_c}}) = \text{LSB}(C^{v_c})$ . The output will not be influenced by or conditioned on the evaluator's randomness or input in any way other than adding a NOT gate to the output; thus, there will not be a selective failure attack.

Therefore, the main task is to check the integrity of the BitGC. It is very challenging for the evaluator to check the correctness because only the garbler holds both labels and defines their values. In particular, during a checking phase, the garbler can provide two different labels, effectively flipping the actual meaning of the evaluator's label. A correctness check based on this would open the possibility of a selective failure attack. The evaluator also has encryption of the permutation bit  $\llbracket \pi_a \rrbracket$ , but the underlying encryption scheme is not committing, and the key belongs to the garbler; it is not clear how to utilize it either. For this reason, our first step is to tweak the protocol so that the garbler no longer has  $\Delta$ . This would help with checking as the garbler cannot swap zero and one labels easily in this case. In addition,  $\Delta$  is also the secret key for the encryption scheme; if the garbler does not have the secret key, it cannot open ciphertexts to other values. Note that this deviates from existing garbling literature as the garbler always has both labels to compute the garbled table.

**BitGC with distributed  $\Delta$ .** Our goal is to let the garbler “commit” to the semantic meaning of the evaluator's label before both parties check the correctness of the evaluation; this way, a corrupted garbler cannot adaptively change the value being checked. Our key observation is that when the evaluator holds a bit encryption, namely  $\llbracket \pi_a \rrbracket$ , if the underlying encryption key is instead secretly shared by two parties, then this encryption would essentially become a commitment as well. We describe how a garbler can compute exactly the same information while  $\Delta$  is secretly shared among the two parties.

Our goal is to maintain the invariant that for each wire, the garbler has  $(A^{u_a}, \pi_a)$ ; the evaluator has  $A^{v_a}$  and that both parties have  $\llbracket \pi_a \rrbracket$  encrypted under global key  $\Delta$  that is shared by the two parties as  $\Delta_A$  and  $\Delta_B$  such that  $\Delta_B = \Delta_A + \Delta$ . Here,  $v_a$  refers to the actual truth value of the wire;  $u_a$  refers to the truth value of the wire when all inputs are 0. Now, we proceed to replicate the steps in the original BitGC protocol, but the garbler only has one label.

1. **Compute unfixed garbled table.** This step is the same as before since both parties still have  $\llbracket \pi_a, \pi_b, r_c \rrbracket$
2. **Compute the output wire mask  $\pi_c$ .** In the original protocol, the garbler uses  $A^{\pi_a}, B^{\pi_b}$  and  $z_{0,0}$  to obtain the wire mask, but it can no longer do it. However, notice that

$$\begin{aligned} \text{Eval}(A^{u_a}, B^{u_b}) &= \text{Eval}(A^{v_a}, B^{v_b}) + (-1)^{\pi_c} \cdot (z_{\pi_a \oplus u_a, \pi_b \oplus u_b} - z_{\pi_a \oplus v_a, \pi_b \oplus v_b}) \cdot \Delta \\ &= C^{v_c} + (-1)^{\pi_c} \cdot (g(u_a, u_b) - g(v_a, v_b)) \cdot \Delta \\ &= C^0 + (-1)^{\pi_c} \cdot u_c \cdot \Delta \\ &= C^{u_c}, \end{aligned}$$

for any  $\tau_1, \tau_2$  and  $\tau_3$  as inputs to Eval, omitted for simplicity. Therefore, we can also compute the wire mask  $\pi_c = \text{LSB}(\text{Eval}(A^{u_a}, B^{u_b}, \tilde{\tau}_1, \tilde{\tau}_2, \tilde{\tau}_3)) \oplus u_c$ .

3. **Transmit a single bit.** This step is identical to the semi-honest version.
4. **Obtain output label  $C^{u_c}$ .** Our goal now is to obtain just  $C^{u_c}$ . This can be computed as  $\text{Eval}(A^{u_a}, B^{u_b}, \tau_1, \tau_2, \tau_3)$  due to the equation above.

Intuitively, what is happening is that the garbler is also “evaluating” the garbled circuit it generates but using all-zero as the input to the whole circuit. This allows the garbler to obtain the wire mask of each wire, which is sufficient to construct the GC bits for the subsequent gates.

**Input processing with a distributed  $\Delta$ .** Because the garbler no longer has all input labels, we also need to design a new way to let two parties maintain the invariant for input wires. In particular, the garbler shall receive all 0-labels (e.g.,  $A^0$ ), while the evaluator shall receive labels matching the circuit inputs (e.g.,  $A^{v_a}$  for input  $v_a$ ).

Our first idea is to utilize the distributed decryption property mentioned earlier: for any ciphertext  $\llbracket m \rrbracket$ ,  $\text{Dec}(\Delta_B, \llbracket m \rrbracket) = \text{Dec}(\Delta_A, \llbracket m \rrbracket) + m \cdot \Delta$ . With this property, the following method can be derived: when both parties know the ciphertext  $\llbracket (-1)^{\pi_a} \cdot v_a \rrbracket$ , setting

$$A^0 = \text{Dec}(\Delta_A, \llbracket (-1)^{\pi_a} \cdot v_a \rrbracket) \text{ and } A^{v_a} = \text{Dec}(\Delta_B, \llbracket (-1)^{\pi_a} \cdot v_a \rrbracket)$$

satisfies our requirements based on distributed decryption. However, this leads to a deadlock similar to garbling the gate: generating  $\llbracket (-1)^{\pi_a} \cdot v_a \rrbracket$  requires  $\pi_a$ , while computing  $\pi_a = \text{LSB}(A^0) = \text{LSB}(\text{Dec}(\Delta_A, \llbracket (-1)^{\pi_a} \cdot v_a \rrbracket))$  also depends on  $\llbracket (-1)^{\pi_a} \cdot v_a \rrbracket$ .

To resolve this, both parties can compute the input labels ( $A^0, A^{v_a}$ ) using the same method as deriving output labels during gate garbling. This works because  $\text{Dec}$  is also an ‘‘odd function’’ for ciphertexts (see I), satisfying  $\text{Dec}(\Delta_A, \llbracket v_a \rrbracket) = -\text{Dec}(\Delta_A, -\llbracket v_a \rrbracket)$ . Specifically, to encode input  $v_a$  using uniform random bit ciphertexts  $\llbracket r_a \rrbracket$  and  $\llbracket \hat{r}_a \rrbracket$ , where the garbler knows  $r_a$  and the party providing input  $v_a$  knows  $\hat{r}_a$ , proceed as follows:

1. **Generate the ciphertext  $\llbracket v_a \rrbracket$ .** If  $v_a$  is the evaluator’s input, the evaluator sends  $\sigma_a := v_a \oplus \hat{r}_a$  to the garbler. Otherwise, the garbler sends  $\sigma_a := v_a \oplus \hat{r}_a$  to the evaluator. Then, both parties homomorphically compute  $\llbracket v_a \rrbracket := \llbracket \hat{r}_a \rrbracket \oplus \sigma_a$ .
2. **Compute the input wire mask  $\pi_a$ .** Both parties first compute  $\llbracket (-1)^{r_a} \cdot v_a \rrbracket$  homomorphically as  $(1 - 2 \cdot \llbracket r_a \rrbracket) \cdot \llbracket v_a \rrbracket$ . Using the oddness of  $\text{Dec}$ , the garbler computes

$$\pi_a = \text{LSB}(\text{Dec}(\Delta_A, (-1)^{\pi_a} \cdot \llbracket v_a \rrbracket)) = \text{LSB}(\text{Dec}(\Delta_A, (-1)^{r_a} \cdot \llbracket v_a \rrbracket)) .$$

Then, the garbler sends  $d_a := r_a \oplus \pi_a$  to the evaluator and both parties compute

$$\llbracket (-1)^{\pi_a} \cdot v_a \rrbracket := (-1)^{d_a} \cdot \llbracket (-1)^{r_a} \cdot v_a \rrbracket .$$

3. **Distributed decrypt for input labels ( $A^0, A^{v_a}$ ).** Both parties generate input labels using their  $\Delta_A$  and  $\Delta_B$ :  $A^0 = \text{Dec}(\Delta_A, \llbracket (-1)^{\pi_a} \cdot v_a \rrbracket)$  and  $A^{v_a} = \text{Dec}(\Delta_B, \llbracket (-1)^{\pi_a} \cdot v_a \rrbracket)$ .

### 2.3 Authenticating Distributed BitGC

Now that we have a protocol such that at the end of the protocol, both parties have  $\llbracket \pi_a \rrbracket$ ,  $P_B$  has  $A^{v_a}$  and  $P_A$  has  $A^{u_a}$  for each wire. No party can decrypt the ciphertext, but they can jointly perform decryption in a way such that no party can flip the plaintext.

For each gate  $(a, b, c, g)$ ,  $P_B$  can compute encrypted  $v_c$  from the input wire mask as

$$\llbracket v_c \rrbracket = g(\llbracket \pi_a \rrbracket \oplus \text{LSB}(A^{v_a}), \llbracket \pi_b \rrbracket \oplus \text{LSB}(B^{v_b})),$$

Since  $\text{LSB}(A^{v_a}) = v_a \oplus \pi_a$  and  $\text{LSB}(B^{v_b}) = v_b \oplus \pi_b$ . Similarly,  $P_B$  could also compute it via

$$\llbracket v'_c \rrbracket = \llbracket \pi_c \rrbracket \oplus \text{LSB}(C^{v_c}).$$

If they are the same, then this gate is computed correctly. It’s tempting to let  $P_B$  compute  $\llbracket v_c \rrbracket - \llbracket v'_c \rrbracket$  and use distributed decryption to check if it is zero. However, although it would be secure against  $P_A$ , a malicious



$P_B$  could replace this ciphertext with any other ciphertext that it wants to perform a zero test, leading to a selective failure attack by  $P_B$ .

To avoid  $P_B$  sending the wrong ciphertext, we ask  $P_B$  send  $\llbracket \pi_i \oplus v_i \rrbracket$  to  $P_A$ , by first sending a valid ciphertext of a seed that can be expanded to many encrypted bits  $t_i$ , then sending a bit  $t_i \oplus \pi_i \oplus v_i$  to  $P_A$ . This way, the only way to cheat from  $P_B$  is to flip the bit, which would lead to abort in the check later. Now  $P_A$  also obtains  $\llbracket v_i \rrbracket$  and  $\llbracket v'_i \rrbracket$  just like the above.

To check that these values are all the same, both parties perform a random linear combination of all differences and obtain one value that is supposed to be an encryption of zero:

$$\tau = \left( \sum_i \chi_i \cdot (\llbracket v_i \rrbracket - \llbracket v'_i \rrbracket) \right),$$

where each  $\chi_i \xleftarrow{\$} \mathbb{Z}_p$ . Following the same spirit, we can also check that the input and output decoding are processed correctly in the same equation by extending the above equation.

Now that both parties have a ciphertext that should be an encryption zero if and only if the computation is correct. They can perform distributed decryption using  $\Delta_A$  and  $\Delta_B$ , obtaining two equation values, since  $\text{Dec}(\Delta_B, \llbracket 0 \rrbracket) = \text{Dec}(\Delta_A, \llbracket 0 \rrbracket) + 0 \cdot \Delta$ .

Overall, this protocol's computational and communication costs per gate are about twice that of the original BitGC protocol. The requirement on the underlying extended GSW scheme is the same, meaning that we don't require evaluating on a deeper circuit.

Note that if the underlying SWHE scheme can perform  $L$  extra levels of homomorphic multiplications, the evaluator can skip some of the bit and let the garbler evaluate them instead. For layered circuits, a circuit with  $C$  gates can be divided into  $L$ -depth subcircuits, with a total output size bounded by  $O(C/L)$ . For generic circuits, the bound is  $O(C \log C/L)$ . When  $L = \omega(1)$  for layered circuits or  $L = \omega(\log C)$  for generic circuits, communication remains  $1 + o(1)$  bits per gate, eliminating the need for bootstrapping.

## 2.4 Two-Party Key Generation and Encryption

The remaining issue is that both  $P_A$  and  $P_B$  should perform encryption in our authenticated BitGC protocol. This is straightforward in BitGC [LWYY25], where  $P_A$  holds the secret key  $\Delta$  and encrypts all plaintexts. However, in our authenticated BitGC setting, both parties share the secret key  $\Delta$ , with  $P_A$  holding  $\Delta_A$  and  $P_B$  holding  $\Delta_B = \Delta_A + \Delta$ , and both parties share the secret key  $\Delta$ , perform encryption, and generate key-related ciphertexts to implement the Eval function. We now describe how to implement two-party key generation and encryption in  $\text{poly}(\lambda)$ , where the complexity of the key generation and encryption algorithms depends only on the security parameter  $\lambda$  and is independent of the circuit size.

We take  $P_A$  encrypting the message  $m$  as an example. First,  $P_A$  sends its ciphertext  $(a, b_A = a \cdot \Delta_A + e_A - m)$  to  $P_B$ , while  $P_B$  sends its ciphertext  $(a, b_B = a \cdot \Delta_B + e_B)$  to  $P_A$ , both using the same public value  $a$ . Next, both parties can locally compute ciphertext  $\llbracket m \rrbracket = (a, b = b_B - b_A = a \cdot \Delta + (e_B - e_A) + m)$  using key and message homomorphism. Applying commitment and zero-knowledge proofs ensures the bounds of  $\|b_A - a \cdot \Delta_A\|_\infty$  and  $\|b_B - a \cdot \Delta_B\|_\infty$ . A similar approach can encrypt  $P_B$ 's message  $m$  or the shared message  $m$  with  $P_A$  holding  $m_A$  and  $P_B$  holding  $m_A + m$ .

Another challenge is generating key-related ciphertexts where the message is a non-linear function of the key. We address this using malicious 2PC protocols, such as SPDZ [DPSZ12, DKL<sup>+</sup>13], to generate the result in shared form and the commitment of the shares. Commitment ensures the consistency of the entire protocol. Subsequently, both parties can apply the previously described method to obtain the ciphertext.



### 3 Preliminaries

**Notation.** We will use  $\lambda$  and  $\rho$  to denote the computational and statistical security parameters, respectively. For  $n \in \mathbb{N}$ , let  $[n] \stackrel{\text{def}}{=} \{1, \dots, n\}$ . We use  $x \stackrel{\$}{\leftarrow} \mathcal{S}$  (resp.,  $x \leftarrow \mathcal{D}$ ) to denote sampling  $x$  from a set  $\mathcal{S}$  uniformly at random (resp., according to a distribution  $\mathcal{D}$ ). For a vector  $\mathbf{a}$ , we use  $\mathbf{a}[i]$  to denote the  $i$ -th component of  $\mathbf{a}$ , where  $\mathbf{a}[1]$  is the first component. Let  $\mathcal{R} \stackrel{\text{def}}{=} \mathbb{Z}[X]/(X^n + 1)$  be a polynomial ring with integer coefficients modulo a polynomial  $X^n + 1$ , and  $\mathcal{R}_q \stackrel{\text{def}}{=} \mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[X]/(X^n + 1)$  for a modulus  $q \in \mathbb{N}$ . For a polynomial  $\mathbf{a} \in \mathcal{R}$ , we define  $\text{LSB}(\mathbf{a}) = \mathbf{a}[1] \bmod 2$ . For a vector  $\mathbf{v}$  of length  $n$ , the infinity norm  $\|\mathbf{v}\|_\infty$  is defined as  $\|\mathbf{v}\|_\infty \stackrel{\text{def}}{=} \max_{i \in [n]} |\mathbf{v}[i]|$ . For two vectors  $\mathbf{x}$  and  $\mathbf{y}$ , we use  $\mathbf{x} \approx \mathbf{y}$  to denote that  $\|\mathbf{x} - \mathbf{y}\|_\infty$  is relatively small.

We use  $\text{negl}(\cdot)$  to denote an unspecified negligible function such that  $\text{negl}(\lambda) = o(\lambda^{-c})$  for every constant  $c$ , and  $\text{poly}(\cdot)$  to denote a polynomial function with  $\text{poly}(\lambda) = O(\lambda^c)$  for some constant  $c$ . We denote the rounding function by  $\lfloor \cdot \rfloor : \mathbb{R} \rightarrow \mathbb{Z}$ , which maps  $x \in \mathbb{R}$  to the closest integer  $y \in \mathbb{Z}$ . For integers  $p$  and  $q$  such that  $2 \leq p \leq q$ , the modular rounding function is defined as  $\lfloor \cdot \rfloor_p : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$  that maps  $x \rightarrow \lfloor (p/q) \cdot x \rfloor$ . This definition extends naturally to vectors and matrices over  $\mathbb{Z}_q$ . For two vectors  $\mathbf{x}$  and  $\mathbf{y}$ ,  $\langle \mathbf{x}, \mathbf{y} \rangle$  represents their inner product.

**Boolean circuits.** Following the notation in [WRK17a], a Boolean circuit is represented as a list of gates of the form  $(\alpha, \beta, \gamma, g)$ , where  $\alpha$  and  $\beta$  are input wire indices,  $\gamma$  is output wire index, and  $g \in \{\wedge, \oplus\}$  specifies the gate type. We use  $\mathcal{P}_A$  and  $\mathcal{P}_B$  to denote the two parties executing a 2PC protocol, and w.l.o.g. assume that only  $\mathcal{P}_B$  obtains the output at the end of 2PC execution. Then we use  $\mathcal{I}_A$  (resp.,  $\mathcal{I}_B$ ) to represent the set of circuit-input wire indices corresponding to the  $\mathcal{P}_A$ 's input (resp., the  $\mathcal{P}_B$ 's input),  $\mathcal{O}$  to denote the set of circuit-output wire indices associated with the  $\mathcal{P}_B$ 's output, and  $\mathcal{G}$  to denote the set of output wire indices of all AND and XOR gates. We also use  $N = |\mathcal{I}_A| + |\mathcal{I}_B| + |\mathcal{G}|$  to denote the number of all wires and  $v_i$  to denote the actual bit on the  $i$ -th wire.

#### 3.1 Security Model and Ideal Functionalities

**Security model.** We adopt the standard ideal/real model [Can00, Gol04] to prove security of our 2PC protocol in the presence of a static, malicious adversary. In the *ideal-world* execution, two parties interact with an ideal functionality  $\mathcal{F}$ , and one of them may be corrupted by an ideal-world adversary (a.k.a., *simulator*)  $\mathcal{S}$ . In the *real-world* execution, the parties interact with each other in an execution of protocol  $\Pi$ , and one of them may be corrupted by a real-world adversary  $\mathcal{A}$  (that is often called the adversary for the sake of simplicity). We say that protocol  $\Pi$  securely computes ideal functionality  $\mathcal{F}$ , if the joint distribution of the outputs of honest party and  $\mathcal{A}$  in the real-world execution is computationally indistinguishable from that of the outputs of honest party and  $\mathcal{S}$  in the ideal-world execution. We prove security of our protocol in the  $\mathcal{H}$ -hybrid model, where two parties execute the protocol with real messages, and also have access to a sub-functionality  $\mathcal{H}$ . Our 2PC protocol can also be proven secure in the universal composition (UC) framework [Can01], if all sub-functionalities used in our protocol are instantiated with UC-secure protocols.

We focus on security with abort, i.e., the adversary, who corrupts one party, first receives the output, and then can determine whether the honest party obtains the output. It is standard in the two-party setting. Therefore, the simulator can send abort to the ideal functionality  $\mathcal{F}$  at any point, which instructs  $\mathcal{F}$  to send abort to the parties and abort. This allows the simulator to prevent the honest party obtain the output, after the corrupted party gets its output. We implicitly allow the simulator to send abort to  $\mathcal{F}$  at any point, and omit the corresponding description for the sake of simplicity.

**The coin-tossing functionality.** Our protocol would invoke the standard coin-tossing functionality  $\mathcal{F}_{\text{Rand}}$  shown in Figure 1, which can be securely computed in the random oracle model (see, e.g., [HSS17, YWZ20]).

**Functionality  $\mathcal{F}_{\text{Rand}}$**

This functionality interacts with  $P_A$  and  $P_B$  as follows:

1. Upon receiving  $(\text{Rand}, S)$  from both parties, where  $S$  is an efficiently sampleable set, sample  $r \xleftarrow{\$} S$ , and then output  $r$  to all parties.

Figure 1: The ideal functionality for coin tossing.

**Functionality  $\mathcal{F}_{\text{CPC}}$**

This functionality is parameterized by a polynomial ring  $\mathcal{R}_q$ . It interacts with two parties  $P_A$  and  $P_B$  as well as the adversary as follows.

- **Commit:** Upon receiving  $(\text{Commit}, \mathcal{P}, id, w)$  from a party  $\mathcal{P} \in \{P_A, P_B\}$ , where  $id$  is a fresh identifier, store  $(\text{com}, \mathcal{P}, id, w)$  and output  $(\mathcal{P}, id)$  to both parties.
- **Prove:** Upon receiving  $(\text{Prove}, \mathcal{P}, id_1, \dots, id_\ell, f)$  from  $P_A$  and  $P_B$ , where  $\mathcal{P} \in \{P_A, P_B\}$ ,  $(\text{com}, \mathcal{P}, id_i)$  for all  $i \in [\ell]$  are present in memory and  $f$  is a verification circuit, retrieve  $(\text{com}, \mathcal{P}, id_i, w_i)$  for each  $i \in [\ell]$ ; if  $f(w_1, \dots, w_\ell) = 1$ , output true to both parties, else output false to both parties.
- **Compute:** Upon receiving  $(\text{Compute}, id_1, \dots, id_t, id, \hat{id}, g)$  from  $P_A$  and  $P_B$ , where  $id_i$  for all  $i \in [t]$  are present in memory, both of  $id, \hat{id}$  are fresh identifiers, and  $g$  is an arithmetic circuit, retrieve  $(\text{com}, \star, id_i, w_i)$  for each  $i \in [t]$  and compute  $\mathbf{y} := g(w_1, \dots, w_t) \in \mathcal{R}_q^n$ . Then,
  - If  $P_A$  is honest, sample  $\mathbf{m}_A \xleftarrow{\$} \mathcal{R}_q^n$ . Otherwise, receive  $\mathbf{m}_A \in \mathcal{R}_q^n$  from the adversary. Compute  $\mathbf{m}_B := \mathbf{m}_A + \mathbf{y} \in \mathcal{R}_q^n$ .
  - If  $P_B$  is corrupted, receive  $\mathbf{m}_B \in \mathcal{R}_q^n$  from the adversary and recompute  $\mathbf{m}_A := \mathbf{m}_B - \mathbf{y} \in \mathcal{R}_q^n$ .
  - Store  $(\text{com}, P_A, id, \mathbf{m}_A)$  and  $(\text{com}, P_B, \hat{id}, \mathbf{m}_B)$ . Then output  $\mathbf{m}_A$  to  $P_A$  and  $\mathbf{m}_B$  to  $P_B$ .

Figure 2: The ideal functionality for the “commit-prove-compute” that allows proving and computing on committed values.

**The ideal functionality to capture the “commit-and-prove” paradigm and secure computation of committed values.** We define an ideal functionality  $\mathcal{F}_{\text{CPC}}$  shown in Figure 2 to model security of the “commit-and-prove” paradigm, where a witness is first committed via the Commit command, and then the statement is proven with the witness via the Prove command. To further capture secure computation of committed values,  $\mathcal{F}_{\text{CPC}}$  allows two parties to call the Compute command to securely compute the values committed by  $P_A$  and  $P_B$ , respectively. As a result, both parties can obtain an additive secret sharing on the computational result, and the shares are committed and can be further used in the Prove command. As in prior works [NNOB12, HSS17, WRK17a, WRK17b, KRRW18, YWZ20, CWYY23], we allow the corrupted party to choose its output share.

We can use any constant-round zero-knowledge (ZK) proofs in the “commit-and-prove” framework, e.g., the recent ZK proofs [WYKW21, DIO21, BMRS21, YSWW21, WYY<sup>+</sup>22, DILO22b, BCC<sup>+</sup>24, LXY25] based on vector oblivious linear evaluation (VOLE) [ADI<sup>+</sup>17, BCGI18], to securely realize the Commit and Prove commands. Such VOLE-ZK proofs are practically efficient, and can prove fastly that a secret is located in a small range, which would be used in our protocol. We can use any actively secure 2PC based on secret sharing (SS) such as SPDZ [DPSZ12, DKL<sup>+</sup>13, KOS16, KPR18] to securely realize the Compute command. Since our protocol needs to securely compute only one-level multiplications on committed values, the round complexity of SS-based 2PC is  $O(1)$  in this case. Overall, we obtain a constant-round protocol

with active security to securely compute  $\mathcal{F}_{\text{CPC}}$  for our application.

When combining VOLE-ZK with SPDZ, we are able to obtain a highly efficient protocol to compute  $\mathcal{F}_{\text{CPC}}$  securely, where one can seamlessly and efficiently convert between the commitments in VOLE-ZK and information-theoretic message authentication codes (IT-MACs) in SPDZ. Specifically, VOLE correlations are used as commitments in VOLE-ZK, and can also be viewed as BDOZ-style IT-MACs [BDOZ11]. It is well-known that BDOZ-style IT-MACs can be locally converted into SPDZ-style IT-MACs. Therefore, the commitments (applied in VOLE-ZK) can be straightforwardly used as the inputs equipped with SPDZ-style IT-MACs in SPDZ-like 2PC protocols. For the IT-MACs output by an SPDZ-like 2PC, they can be transformed to BDOZ-style IT-MACs requiring one element per IT-MAC of communication [HOSS18]. Then, these BDOZ-style IT-MACs can be used as the commitments in VOLE-ZK.

### 3.2 Basic Definitions and Lemmas for Lattice-based Cryptography

We summarize the basic definitions and lemmas for SWHE with distributed decryption.

**Definition 1** (Discrete Gaussian Distribution). *The discrete Gaussian distribution  $D_\sigma^m$  over  $\mathbb{Z}^m$  is centered at  $\mathbf{0}$  with each coordinate sampled independently and having a standard deviation of  $\sigma$ . For any  $\mathbf{x} \in \mathbb{Z}^m$ , the discrete Gaussian distribution is defined as  $D_\sigma^m(\mathbf{x}) = \rho_\sigma^m(\mathbf{x}) / \rho_\sigma^m(\mathbb{Z}^m)$ , where*

$$\rho_\sigma^m(\mathbf{x}) = \prod_{i=1}^m \left( \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x[i]^2}{2\sigma^2}} \right)$$

is the continuous Gaussian function over  $\mathbb{R}^m$  and  $\rho_\sigma^m(\mathbb{Z}^m) = \sum_{\mathbf{z} \in \mathbb{Z}^m} \rho_\sigma^m(\mathbf{z})$  is the normalization factor ensuring the probabilities sum to 1.

Since a ring element in  $\mathcal{R}$  can be viewed as a vector in  $\mathbb{Z}^n$ , we use  $\mathcal{D}_\eta$  to denote a discrete Gaussian distribution over  $\mathcal{R}$ , such that  $\Pr_{e \leftarrow \mathcal{D}_\eta} [\|e\|_\infty > \eta]$  is negligible in  $\lambda$ , where  $\eta$  is a parameter. Besides, we use  $\mathcal{D}_\eta^N$  to denote a distribution of vectors with each component sampling from  $\mathcal{D}_\eta$ .

**Lemma 1** (Lifting Lemma [BKS19]). *Let  $p \in \mathbb{N}$  be a modulus with  $p \geq n^{\omega(1)}$  and let  $z_0 \in \mathcal{R}$  be a uniformly random ring element with coefficients in  $\mathbb{Z}_p$ . For any  $m \in \mathcal{R}$ , let  $z_1 = z_0 + m \pmod p$ . Then, we have*

$$\Pr [z_1 = z_0 + m] \geq 1 - n \cdot (\|m\|_\infty + 1) / p.$$

**Lemma 2** (Rounding Lemma [BKS19]). *Let  $p, q \in \mathbb{N}$  be two moduli with  $q/p \geq n^{\omega(1)}$  and  $p \mid q$ . Let  $t_0 \in \mathcal{R}_q$  be a uniformly random ring element, and define  $t_1 = t_0 + (q/p) \cdot m + e$  over  $\mathcal{R}_q$  for some  $m \in \mathcal{R}_p$  and  $e \in \mathcal{R}$ . Then, we have*

$$\Pr [\lfloor t_1 \rfloor_p = \lfloor t_0 \rfloor_p + m] \geq 1 - n \cdot (\|e\|_\infty + 1) \cdot p/q.$$

The ring learning with errors (RLWE) problem, introduced by Lyubashevsky, Peikert, and Regev [LPR10], is defined as follows.

**Definition 2** (RLWE [LPR10]). *For dimension  $n \in \mathbb{N}$ , number of samples  $m \in \mathbb{N}$ , and modulus  $q \in \mathbb{N}$ , the RLWE( $n, m, \eta, q$ ) problem is to distinguish the following two distributions:*

$$\left\{ \left( \mathbf{a}_i \xleftarrow{\$} \mathcal{R}_q, \mathbf{b}_i = \mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i \right) \right\}_{i \in [m]} \quad \text{and} \quad \left\{ \left( \mathbf{a}_i \xleftarrow{\$} \mathcal{R}_q, \mathbf{u}_i \xleftarrow{\$} \mathcal{R}_q \right) \right\}_{i \in [m]},$$

where  $\mathbf{s} \leftarrow \mathcal{D}_\eta$  and  $\mathbf{e}_i \leftarrow \mathcal{D}_\eta$ , for each  $i \in [m]$ . The RLWE( $n, m, \eta, q$ ) assumption is that the RLWE( $n, m, \eta, q$ ) problem is infeasible.

$\mathcal{D}_\eta$  is defined as above, and RLWE also supports other error distributions. For the sake of simplicity, we write the RLWE assumption or RLWE problem, when the parameters are clear from the context.

### 3.3 SWHE with Distributed Decryption

We recall the recent notion of somewhat homomorphic encryption (SWHE) with distributed decryption. The syntax is described in the following definition. We also describe the RLWE-based instantiation of such SWHE scheme in Section 5.1, following prior work [LWYY25].

**Definition 3** (SWHE with Distributed Decryption [LWYY25]). *A somewhat homomorphic encryption scheme with distributed decryption in the private-key setting consists of the following polynomial-time algorithms and satisfies the message homomorphism and distributed decryption properties. Let  $\mathcal{K}$  be the space of secret keys,  $\mathcal{R}_p$  be the space of key shares,  $\mathcal{M} \subseteq \mathcal{R}_p$  be the message space, and  $\mathcal{C}$  be the ciphertext space.*

- **Setup:**  $\text{params} \leftarrow \text{Setup}(1^\lambda, 1^L)$ . *The setup algorithm takes as input a security parameter  $\lambda$  and a maximum multiplication depth  $L$ , and then outputs a set of parameters  $\text{params}$  that is independent of secret keys. Here,  $\text{params}$  is an implicit input to the following encryption and decryption algorithms, and would be omitted for simplicity.*
- **Key Generation:**  $(\text{keypar}, \text{sk}) \leftarrow \text{KeyGen}(\text{params})$ . *The key generation algorithm takes as input  $\text{params}$ , and then outputs a set of key-related parameters  $\text{keypar}$ , along with a secret key  $\text{sk} \in \mathcal{K}$  such that  $\text{LSB}(\text{sk}) = 1$ . Similarly,  $\text{keypar}$  is an implicit input to the following algorithms and is omitted for simplicity.*
- **Encryption:**  $\llbracket m \rrbracket \leftarrow \text{Enc}(\text{sk}, m)$ . *The encryption algorithm takes as input a secret key  $\text{sk} \in \mathcal{K}$  and a message  $m \in \mathcal{M}$ , and outputs a ciphertext  $\llbracket m \rrbracket \in \mathcal{C}$ .*
- **Decryption:**  $m \cdot \text{sk} \leftarrow \text{Dec}(\text{sk}, \llbracket m \rrbracket)$ . *The decryption algorithm takes as input a secret key  $\text{sk}$  and a ciphertext  $\llbracket m \rrbracket$ , and outputs  $m \cdot \text{sk}$  over  $\mathcal{R}_p$ . It is w.l.o.g. assumed that  $m$  can be recovered from  $m \cdot \text{sk}$  over  $\mathcal{R}_p$  with secret key  $\text{sk}$ .*

The SWHE scheme needs to satisfy the following properties.

1. **Message homomorphism.** *For any  $\text{params} \leftarrow \text{Setup}(1^\lambda, 1^L)$ ,  $(\text{keypar}, \text{sk}) \leftarrow \text{KeyGen}(\text{params})$ , message  $m_i \in \mathcal{M}$  and ciphertext  $\llbracket m_i \rrbracket = \text{Enc}(\text{sk}, m_i)$ , for each  $i \in [\ell]$ , where integer  $\ell \geq 1$ , the following equality holds with probability  $1 - \text{negl}(\lambda)$ , for any polynomial-sized circuit  $f$  with a low depth  $L$ :*

$$\text{Dec}(\text{sk}, \tilde{f}(\llbracket m_1 \rrbracket, \dots, \llbracket m_\ell \rrbracket)) = f(m_1, \dots, m_\ell) \cdot \text{sk},$$

where  $\tilde{f}$  denotes the homomorphic evaluation of circuit  $f$  on the ciphertexts, which can be performed in time  $\text{poly}(\lambda)$ .

2. **Distributed decryption.** *For each  $\text{params} \leftarrow \text{Setup}(1^\lambda, 1^L)$ ,  $(\text{keypar}, \text{sk}) \leftarrow \text{KeyGen}(\text{params})$  and ciphertext  $\llbracket m \rrbracket = \text{Enc}(\text{sk}, m)$  on any message  $m \in \mathcal{M}$ , the following properties hold:*

- **Linear distributed decryption.** *Let  $\text{sk}_0$  be an element sampled uniformly from  $\mathcal{R}_p$  such that  $\text{LSB}(\text{sk}_0) = 0$ . Let  $\text{sk}_1 = \text{sk}_0 + \text{sk}$  over  $\mathcal{R}_p$ . Then, the following equations hold with probability  $1 - \text{negl}(\lambda)$ ,*

$$\begin{aligned} \text{Dec}(\text{sk}_i, \llbracket m \rrbracket) &= -\text{Dec}(\text{sk}_i, -\llbracket m \rrbracket) \text{ for } i \in \{0, 1\} \\ \text{Dec}(\text{sk}_1, \llbracket m \rrbracket) &= \text{Dec}(\text{sk}_0, \llbracket m \rrbracket) + \text{Dec}(\text{sk}, \llbracket m \rrbracket). \end{aligned}$$

*The two equations described as above imply  $\text{Dec}(\text{sk}, \llbracket m \rrbracket) = -\text{Dec}(\text{sk}, -\llbracket m \rrbracket)$ .*

- **Correlated-key distributed decryption.** Let  $sk_0, sk'_0$  be two elements sampled uniformly from  $\mathcal{R}_p$  such that  $\text{LSB}(sk_0) = 0$  and  $\text{LSB}(sk'_0) = 0$ . Let  $sk_1 = sk_0 + sk$  and  $sk'_1 = sk'_0 + sk$  over  $\mathcal{R}_p$ . There exists a polynomial-time algorithm  $\widehat{\text{Dec}}$  such that the following equations hold with probability  $1 - \text{negl}(\lambda)$ ,

$$\begin{aligned}\widehat{\text{Dec}}(sk_0, sk'_0, \llbracket m \rrbracket) &= -\widehat{\text{Dec}}(sk_0, sk'_0, -\llbracket m \rrbracket), \\ \widehat{\text{Dec}}(sk_i, sk'_j, \llbracket m \rrbracket) - \widehat{\text{Dec}}(sk_0, sk'_0, \llbracket m \rrbracket) &= i \cdot j \cdot \text{Dec}(sk, \llbracket m \rrbracket) \text{ for } i, j \in \{0, 1\}.\end{aligned}$$

For the sake of simplifying the description, we define a polynomial-time algorithm  $\text{Eval}$  using the above algorithms  $\text{Dec}$  and  $\widehat{\text{Dec}}$ . Specifically, given the ciphertexts  $\tau_1, \tau_2, \tau_3$  encrypted under  $sk$ , for any  $X, Y \in \mathcal{R}_p$ ,  $\text{Eval}$  is defined as:

$$\text{Eval}(X, Y, \tau_1, \tau_2, \tau_3) \stackrel{\text{def}}{=} \text{Dec}(X, \tau_1) + \text{Dec}(Y, \tau_2) + \widehat{\text{Dec}}(X, Y, \tau_3).$$

For any ciphertexts  $\tau_1, \tau_2, \tau_3$  and uniform elements  $X, Y \in \mathcal{R}_p$  with  $\text{LSB}(X) = \text{LSB}(Y) = 0$ , with probability  $1 - \text{negl}(\lambda)$ , we have

$$\begin{aligned}\text{Eval}(X, Y, -\tau_1, -\tau_2, -\tau_3) &= \text{Dec}(X, -\tau_1) + \text{Dec}(Y, -\tau_2) + \widehat{\text{Dec}}(X, Y, -\tau_3) \\ &= - \left( \text{Dec}(X, \tau_1) + \text{Dec}(Y, \tau_2) + \widehat{\text{Dec}}(X, Y, \tau_3) \right) \\ &= - \text{Eval}(X, Y, \tau_1, \tau_2, \tau_3).\end{aligned}$$

Furthermore, for any ciphertexts  $\tau_1, \tau_2, \tau_3$  and uniform elements  $X, Y \in \mathcal{R}_p$  with  $\text{LSB}(X) = \text{LSB}(Y) = 0$ , for  $i, j \in \{0, 1\}$ , with probability  $1 - \text{negl}(\lambda)$ , we have

$$\begin{aligned}\text{Eval}(X + i \cdot sk, Y + j \cdot sk, \tau_1, \tau_2, \tau_3) - \text{Eval}(X, Y, \tau_1, \tau_2, \tau_3) &= \text{Dec}(X + i \cdot sk, \tau_1) + \text{Dec}(Y + j \cdot sk, \tau_2) + \widehat{\text{Dec}}(X + i \cdot sk, Y + j \cdot sk, \tau_3) \\ &\quad - \left( \text{Dec}(X, \tau_1) + \text{Dec}(Y, \tau_2) + \widehat{\text{Dec}}(X, Y, \tau_3) \right) \\ &= i \cdot \text{Dec}(sk, \tau_1) + j \cdot \text{Dec}(sk, \tau_2) + i \cdot j \cdot \text{Dec}(sk, \tau_3).\end{aligned}$$

As in [LWYY25], we need that the SWHE scheme satisfies the standard IND-CPA security (i.e., indistinguishability under chosen-plaintext attacks).

## 4 Authenticated BitGC: Constant-Round 2PC with Active Security

We first define an ideal functionality  $\mathcal{F}_{2\text{PSWHE}}$  (Figure 3) to model the security of key generation and encryption in the two-party setting for SWHE with distributed decryption. In this functionality, an honest party always samples either  $\Delta_A \leftarrow \mathcal{D}_\eta$  or  $\Delta_B \leftarrow \mathcal{D}_\eta$ , and inputs it to  $\mathcal{F}_{2\text{PSWHE}}$ . While other distributions are allowed for secret keys, we choose  $\mathcal{D}_\eta$  for simplicity. The  $\text{GenKey}$  command needs to be called only once, and can be reused for multiple  $\text{EncBit}$  commands. If  $\mathcal{P}$  wants to encrypt a vector  $z$  and it is corrupted, we allow the adversary to choose the randomness  $r$  for encryption. It has no impact on security, as  $\mathcal{P}$  owning the plaintext  $z$  has been corrupted.

We present an actively secure two-party protocol to instantiate  $\mathcal{F}_{2\text{PSWHE}}$  in Section 5. To make the protocol concretely efficient, e.g., avoiding to running a generic 2PC protocol to sample elements from  $\mathcal{D}_\eta$  which is prohibitively expensive, we define an alternative key-generation algorithm  $\text{KeyGen}'$  for SWHE. In particular,  $\text{KeyGen}'$  takes as input params,  $\Delta_A \in \mathcal{R}$  and  $\Delta_B \in \mathcal{R}$ , and then outputs a set of key-dependent parameters  $\text{keypar}'$  as well as a secret key  $\Delta$ , such that  $\Delta = \Delta_B - \Delta_A$ . Here, we require that  $\|\Delta_A\|_\infty \leq \eta$ ,

### Functionality $\mathcal{F}_{2\text{PSWHE}}$

Let  $\text{KeyGen}'$  (resp.,  $\text{Enc}$ ) be the alternative key-generation algorithm (resp., the encryption algorithm) of an SWHE scheme. This functionality is parameterized by a set of parameters  $\text{params} = \text{Setup}(1^\lambda, 1^L)$ , which defines a polynomial ring  $\mathcal{R}_p$  and a parameter  $\eta \ll p$ . It interacts with two parties  $P_A, P_B$  as follows.

- Upon receiving  $(\text{GenKey}, \Delta_A)$  from  $P_A$  and  $(\text{GenKey}, \Delta_B)$  from  $P_B$ , where  $\Delta_A, \Delta_B \in \mathcal{R}_p$ , do the following:
  1. Check  $\|\Delta_A\|_\infty \leq \eta$ ,  $\|\Delta_B\|_\infty \leq \eta$ ,  $\Delta_A[1] = 0$  and  $\Delta_B[1] = 1$ . If the check fails, send abort to both parties and abort.
  2. Run  $(\text{keypar}', \Delta) \leftarrow \text{KeyGen}'(\text{params}, \Delta_A, \Delta_B)$ , where  $\Delta_B - \Delta_A = \Delta \in \mathcal{R}_p$ ,  $\|\Delta\|_\infty \leq 2\eta$  and  $\text{LSB}(\Delta) = 1$ .
  3. Output  $\text{keypar}'$  to both parties.

Ignore any subsequent  $\text{GenKey}$  commands.

- Upon receiving  $(\text{EncBit}, t, z)$  from a party  $\mathcal{P} \in \{P_A, P_B\}$  and  $(\text{EncBit}, t)$  from the other party, where  $t \in \mathbb{N}$  and  $z \in \{0, 1\}^t$ , do the following:
  1. If  $\mathcal{P}$  is corrupted, receive  $r$  from the adversary, and then run  $\llbracket z_i \rrbracket \leftarrow \text{Enc}(\Delta, z_i; r)$  where  $r$  is the randomness used in the encryption.
  2. If  $\mathcal{P}$  is honest, for each  $i \in [t]$ , run  $\llbracket z_i \rrbracket \leftarrow \text{Enc}(\Delta, z_i)$ , and then set  $\llbracket z \rrbracket := (\llbracket z_1 \rrbracket, \dots, \llbracket z_t \rrbracket)$ .
  3. Output  $\llbracket z \rrbracket$  to both parties.

Figure 3: The ideal functionality for two-party key generation and encryption on an SWHE scheme with distributed decryption.

$\|\Delta_B\|_\infty \leq \eta$ ,  $\Delta_A[1] = 0$  and  $\Delta_B[1] = 1$  for some parameter  $\eta$ . Thus, we have  $\Delta \in \mathcal{R}$ ,  $\|\Delta\|_\infty \leq 2\eta$  and  $\text{LSB}(\Delta) = 1$ . We implicitly assume that both message-homomorphic and distributed-decryption properties still hold under any pair  $(\text{keypar}', \Delta) \leftarrow \text{KeyGen}'(\text{params}, \Delta_A, \Delta_B)$  with  $\Delta_A, \Delta_B$  satisfying the above conditions. This assumption naturally holds for our protocol shown in Section 5.2. Furthermore, we require that the IND-CPA security is still satisfied under any  $(\text{keypar}', \Delta) \leftarrow \text{KeyGen}'(\text{params}, \Delta_A, \Delta_B)$ , if at least one of  $\Delta_A$  and  $\Delta_B$  is sampled from  $\mathcal{D}_\eta$ . The instantiation of  $\text{KeyGen}'$  is given in Section 5.2, and we prove that the SWHE scheme with  $(\text{keypar}', \Delta) = \text{KeyGen}'(\text{params}, \Delta_A, \Delta_B)$  is IND-CPA secure.

Below, we present the details of our authenticated BitGC protocol in Section 4.1. Then, we analyze its correctness based on the passively secure BitGC protocol [LWYY25] in Section 4.2. Finally, we formally prove its security against malicious adversaries in Section 4.3.

#### 4.1 Details of Our Authenticated BitGC Protocol

The details of our authenticated BitGC protocol in the  $(\mathcal{F}_{2\text{PSWHE}}, \mathcal{F}_{\text{Rand}})$ -hybrid model are shown in Figures 4 and 5. In this protocol,  $\text{PRG} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{|\mathcal{I}_A|+N}$  denotes a pseudorandom generator, and  $\text{PRF} : \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \mathcal{R}_p$  represents a pseudorandom function, where  $\mathcal{R}_p$  is defined by a set of parameters  $\text{params}$ . As in prior work [LWYY25], we consider that  $p$  is an even. We divide the protocol into two phases: the preprocessing phase (where the circuit and inputs are unknown) and online phase (where the circuit along with two parties' inputs are known).

Following prior works [BKS19, LWYY25], we use PRF and a random key to generate common elements in  $\mathcal{R}_p$  and then mask the labels, which make the distributed-decryption property (defined in Section 3) hold. We also randomize the key  $\Delta_A$  with a random element  $U \in \mathcal{R}_p$  output by  $\mathcal{F}_{\text{Rand}}$ , which is useful

**Protocol  $\Pi_{2PC}$  (Part 1)**

**Inputs:**  $P_A$  and  $P_B$  agree on a Boolean circuit  $f : \{0, 1\}^{|\mathcal{I}_A|} \times \{0, 1\}^{|\mathcal{I}_B|} \rightarrow \{0, 1\}^{|\mathcal{O}|}$  with  $|\mathcal{I}_A| = |\mathcal{I}_B|$  and a set of parameters  $\text{params} = \text{Setup}(1^\lambda, 1^L)$ .  $P_A$  holds an input  $\mathbf{x} = (x_i)_{i \in \mathcal{I}_A} \in \{0, 1\}^{|\mathcal{I}_A|}$ ;  $P_B$  holds an input  $\mathbf{y} = (y_i)_{i \in \mathcal{I}_B} \in \{0, 1\}^{|\mathcal{I}_B|}$ . Let Dec and Eval be the algorithms defined in SWHE with distributed decryption. Let  $H : \mathcal{R}_p \rightarrow \{0, 1\}^\lambda$  be a cryptographic hash function modeled as a random oracle.

**Preprocessing:** In the preprocessing phase, the circuit and input are unknown, and only  $N = |\mathcal{I}_A| + |\mathcal{I}_B| + |\mathcal{G}|$  is known.  $P_A$  and  $P_B$  execute as follows:

1.  $P_A$  samples  $\Delta_A \leftarrow \mathcal{D}_\eta$ ;  $P_B$  samples  $\Delta_B \leftarrow \mathcal{D}_\eta$ . Then,  $P_A$  and  $P_B$  call the GenKey command of  $\mathcal{F}_{2PSWHE}$  on respective inputs  $\Delta_A$  and  $\Delta_B$  to let both parties obtain a set of key-dependent parameters  $\text{keypar}'$ .
2.  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{Rand}}$  to obtain a random PRF key  $\mathbf{k} \in \{0, 1\}^\lambda$  and a random element  $U \in \mathcal{R}_p$  such that  $\text{LSB}(U) = 0$ . Then,  $P_A$  updates  $\Delta_A := \Delta_A + U \in \mathcal{R}_p$ , and  $P_B$  updates  $\Delta_B := \Delta_B + U \in \mathcal{R}_p$ .
3.  $P_A$  samples  $\mathbf{s}_A \xleftarrow{\$} \{0, 1\}^\lambda$  and computes  $\{\widehat{r}_i\}_{i \in \mathcal{I}_A} \cup \{r_i\}_{i \in [N]} := \text{PRG}(\mathbf{s}_A)$ .  $P_A$  and  $P_B$  call  $\mathcal{F}_{2PSWHE}$  on the  $P_A$ 's input (EncBit,  $\lambda$ ,  $\mathbf{s}_A$ ) and  $P_B$ 's input (EncBit,  $\lambda$ ) to let both parties obtain a vector of ciphertexts  $\llbracket \mathbf{s}_A \rrbracket$ . Then,  $P_A$  and  $P_B$  homomorphically evaluate  $\text{PRG}(\llbracket \mathbf{s}_A \rrbracket)$  to obtain  $\{\llbracket \widehat{r}_i \rrbracket\}_{i \in \mathcal{I}_A}$  and  $\{\llbracket r_i \rrbracket\}_{i \in [N]}$ .
4.  $P_B$  samples  $\mathbf{s}_B \xleftarrow{\$} \{0, 1\}^\lambda$  and computes  $\{\widehat{t}_i\}_{i \in \mathcal{I}_B} \cup \{t_i\}_{i \in [N]} := \text{PRG}(\mathbf{s}_B)$ .  $P_A$  and  $P_B$  call  $\mathcal{F}_{2PSWHE}$  on the  $P_A$ 's input (EncBit,  $\lambda$ ) and  $P_B$ 's input (EncBit,  $\lambda$ ,  $\mathbf{s}_B$ ) to let both parties obtain  $\llbracket \mathbf{s}_B \rrbracket$ . Then,  $P_A$  and  $P_B$  homomorphically evaluate  $\text{PRG}(\llbracket \mathbf{s}_B \rrbracket)$  to obtain  $\{\llbracket \widehat{t}_i \rrbracket\}_{i \in \mathcal{I}_B}$  and  $\{\llbracket t_i \rrbracket\}_{i \in [N]}$ .

**Input processing:** In the online phase, the circuit  $f$  along with the inputs  $\mathbf{x}$  and  $\mathbf{y}$  are known.  $P_A$  and  $P_B$  do the following:

5. For each  $i \in \mathcal{I}_A$ ,  $P_A$  sends  $\sigma_{i,A} := x_i \oplus \widehat{r}_i \in \{0, 1\}$  to  $P_B$ ; both parties homomorphically compute  $\llbracket x_i \rrbracket := \llbracket \widehat{r}_i \rrbracket \oplus \sigma_{i,A}$ .
6. For each  $i \in \mathcal{I}_B$ ,  $P_B$  sends  $\sigma_{i,B} := y_i \oplus \widehat{t}_i \in \{0, 1\}$  to  $P_A$ ; both parties homomorphically compute  $\llbracket y_i \rrbracket := \llbracket \widehat{t}_i \rrbracket \oplus \sigma_{i,B}$ .
7. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B$  and each ciphertext  $\llbracket v_i^* \rrbracket \in \{\llbracket x_i \rrbracket\}_{i \in \mathcal{I}_A} \cup \{\llbracket y_i \rrbracket\}_{i \in \mathcal{I}_B}$ ,
  - (a)  $P_A$  and  $P_B$  homomorphically compute  $\llbracket (-1)^{r_i} \cdot v_i^* \rrbracket := (1 - 2 \cdot \llbracket r_i \rrbracket) \cdot \llbracket v_i^* \rrbracket$ .
  - (b)  $P_A$  computes the following:

$$\widetilde{W}_{i,A} := \text{Dec}(\Delta_A, \llbracket (-1)^{r_i} \cdot v_i^* \rrbracket) + \text{PRF}(\mathbf{k}, i) \in \mathcal{R}_p \text{ and } \pi_i := \text{LSB}(\widetilde{W}_{i,A}).$$

- (c)  $P_A$  sends a bit  $d_i := r_i \oplus \pi_i$  to  $P_B$ . Then,  $P_A$  and  $P_B$  compute

$$\llbracket (-1)^{\pi_i} \cdot v_i^* \rrbracket := (-1)^{d_i} \cdot \llbracket (-1)^{r_i} \cdot v_i^* \rrbracket.$$

- (d)  $P_A$  homomorphically computes  $\llbracket \pi_i \rrbracket := \llbracket r_i \rrbracket \oplus d_i$ .

- (e)  $P_A$  and  $P_B$  compute the following labels respectively:

$$\begin{aligned} W_{i,A} &:= \text{Dec}(\Delta_A, \llbracket (-1)^{\pi_i} \cdot v_i^* \rrbracket) + \text{PRF}(\mathbf{k}, i) \in \mathcal{R}_p, \\ W_{i,B} &:= \text{Dec}(\Delta_B, \llbracket (-1)^{\pi_i} \cdot v_i^* \rrbracket) + \text{PRF}(\mathbf{k}, i) \in \mathcal{R}_p. \end{aligned}$$

Figure 4: Our constant-round 2PC with active security in the  $(\mathcal{F}_{2PSWHE}, \mathcal{F}_{\text{Rand}})$ -hybrid model (Part 1).



**Protocol  $\Pi_{2PC}$  (Part 2)**

**Circuit garbling:**  $P_A$  garbles the circuit as follows.

8. For each gate  $(a, b, c, g)$  in topological order, given a pair of input labels  $(W_{a,A}, W_{b,A}) \in \mathcal{R}_p^2$ , a pair of wire masks  $(\pi_a, \pi_b)$ , and ciphertexts  $(\llbracket \pi_a \rrbracket, \llbracket \pi_b \rrbracket, \llbracket r_c \rrbracket)$ ,  $P_A$  performs the following steps:

(a) Homomorphically compute

$$\begin{aligned} \llbracket (-1)^{r_c} \cdot z_{i,j} \rrbracket &:= (1 - 2 \cdot \llbracket r_c \rrbracket) \cdot g(\llbracket \pi_a \rrbracket \oplus i, \llbracket \pi_b \rrbracket \oplus j) \text{ for each } i, j \in \{0, 1\}, \\ \tilde{\tau}_1 &:= \llbracket (-1)^{r_c} \cdot z_{1,0} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket, \\ \tilde{\tau}_2 &:= \llbracket (-1)^{r_c} \cdot z_{0,1} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket, \\ \tilde{\tau}_3 &:= \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket + \llbracket (-1)^{r_c} \cdot z_{1,1} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{1,0} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,1} \rrbracket, \end{aligned}$$

where  $z_{i,j} = g(\pi_a \oplus i, \pi_b \oplus j) \in \{0, 1\}$  for  $i, j \in \{0, 1\}$ .

(b) Compute  $\widetilde{W}_{c,A} := \text{Eval}(W_{a,A}, W_{b,A}, \tilde{\tau}_1, \tilde{\tau}_2, \tilde{\tau}_3) + \text{PRF}(\mathbf{k}, c)$ . Then compute  $\alpha := \text{LSB}(W_{a,A})$ ,  $\beta := \text{LSB}(W_{b,A})$ , and  $\pi_c := \text{LSB}(\widetilde{W}_{c,A}) \oplus g(\pi_a \oplus \alpha, \pi_b \oplus \beta)$ . Set a bit  $d_c := r_c \oplus \pi_c$ , and homomorphically compute  $\llbracket \pi_c \rrbracket := \llbracket r_c \rrbracket \oplus d_c$ .

(c) Compute  $\tau_i := (-1)^{d_c} \cdot \tilde{\tau}_i$  for each  $i \in \{1, 2, 3\}$ .

(d) Compute  $W_{c,A} := \text{Eval}(W_{a,A}, W_{b,A}, \tau_1, \tau_2, \tau_3) + \text{PRF}(\mathbf{k}, c)$ .

9.  $P_A$  sends a set of bits  $\{d_i\}_{i \in G}$  to  $P_B$ .

**Circuit evaluating:**  $P_B$  evaluates the circuit as follows.

10. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ ,  $P_B$  homomorphically computes  $\llbracket \pi_i \rrbracket := \llbracket r_i \rrbracket \oplus d_i$ .

11. For each gate  $(a, b, c, g)$  in topological order, given a pair of input labels  $(W_{a,B}, W_{b,B}) \in \mathcal{R}_p^2$  and ciphertexts  $(\llbracket \pi_a \rrbracket, \llbracket \pi_b \rrbracket, \llbracket r_c \rrbracket)$ ,  $P_B$  executes as follows:

(a) Perform Step 8a and Step 8c to obtain ciphertexts  $\tau_1, \tau_2, \tau_3$ .

(b) Compute  $W_{c,B} := \text{Eval}(W_{a,B}, W_{b,B}, \tau_1, \tau_2, \tau_3) + \text{PRF}(\mathbf{k}, c)$ .

12. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ ,  $P_B$  sends a bit  $\tilde{d}_i := t_i \oplus \text{LSB}(W_{i,B})$  to  $P_A$ .

**Output processing and consistency check:**  $P_A$  and  $P_B$  execute as follows.

13. For each  $i \in O$ ,  $P_A$  sends a bit  $\pi_i$  to  $P_B$ , who computes  $v_i := \text{LSB}(W_{i,B}) \oplus \pi_i$ .

14. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ ,  $P_A$  and  $P_B$  homomorphically compute  $\llbracket v_i \rrbracket := \llbracket \pi_i \rrbracket \oplus \llbracket t_i \rrbracket \oplus \tilde{d}_i$ .

15. For each  $i \in \mathcal{I}_A$  (resp.,  $i \in \mathcal{I}_B$ ),  $P_A$  and  $P_B$  set  $\llbracket v_i^* \rrbracket := \llbracket x_i \rrbracket$  (resp.,  $\llbracket v_i^* \rrbracket := \llbracket y_i \rrbracket$ ). For each gate  $(a, b, c, g)$ ,  $P_A$  and  $P_B$  homomorphically compute  $\llbracket v_c^* \rrbracket := g(\llbracket v_a \rrbracket, \llbracket v_b \rrbracket)$ .

16.  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{Rand}}$  to generate random coefficients  $\chi_i \in \mathbb{Z}_p$  for  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$  and  $\chi'_i \in \mathbb{Z}_p$  for  $i \in O$ . Then,  $P_A$  and  $P_B$  homomorphically compute

$$\tau := \sum_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G} \chi_i \cdot (\llbracket v_i^* \rrbracket - \llbracket v_i \rrbracket) + \sum_{i \in O} \chi'_i \cdot (\llbracket \pi_i \rrbracket - \pi_i).$$

17.  $P_A$  sends  $V_A := H(\text{Dec}(\Delta_A, \tau), \tau)$  to  $P_B$ , who computes  $V_B := H(\text{Dec}(\Delta_B, \tau), \tau)$  and checks  $V_A = V_B$ . If the check passes, then  $P_B$  outputs  $\{v_i\}_{i \in O}$ . Otherwise,  $P_B$  aborts.

Figure 5: Our constant-round 2PC with active security in the  $(\mathcal{F}_{2PSWHE}, \mathcal{F}_{\text{Rand}})$ -hybrid model (Part 2).

to adopt the distributed-decryption property for  $\Delta_B = \Delta_A + \Delta \in \mathcal{R}_p$ . Since  $\text{LSB}(U) = 0$ , we have  $\text{LSB}(\Delta_A + U) = \text{LSB}(\Delta_A) = 0$  and  $\text{LSB}(\Delta_B + U) = \text{LSB}(\Delta_B) = 1$ . After the randomization, we still have  $\Delta_B = \Delta_A + \Delta$  over  $\mathcal{R}_p$ .

**Analysis for communication, computation and round complexities.** In the following, we analyze the communication and computation costs as well as rounds for protocol  $\Pi_{2\text{PC}}$  (Figures 4 and 5) in the  $(\mathcal{F}_{2\text{PSWHE}}, \mathcal{F}_{\text{Rand}})$ -hybrid model.

There is no communication for the preprocessing phase in the hybrid model. When  $\mathcal{F}_{2\text{PSWHE}}$  and  $\mathcal{F}_{\text{Rand}}$  are instantiated, the communication cost in the preprocessing phase is independent of the circuit size, and can be negligibly small for sufficiently large circuits. In the online phase, the communication is to transmit the bits  $\{\sigma_{i,A}\}_{i \in \mathcal{I}_A}$ ,  $\{\sigma_{i,B}\}_{i \in \mathcal{I}_B}$ ,  $\{d_i\}_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}}$ ,  $\{\tilde{d}_i\}_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}}$ ,  $\{\pi_i\}_{i \in \mathcal{O}}$  along with the  $\lambda$ -bit string  $V_A$ . Thus, the total communication cost is  $3(|\mathcal{I}_A| + |\mathcal{I}_B|) + 2C + |\mathcal{O}| + \lambda$  bits where  $C = |\mathcal{G}|$  is the number of all gates. In other words, the total communication is dominated by 2 bits per gate.

The computational cost is dominated by the depth of homomorphic multiplications and the number of homomorphic operations for SWHE. Using a low-depth PRG, e.g., LWR-based PRG [BPR12], Goldreich-like PRG [App12], and LPN-based weak PRF [BIP<sup>+</sup>18, DGH<sup>+</sup>21, APRR24] (which can be converted into a PRG using a random oracle), the homomorphic-multiplication depth for SWHE can be bounded by  $O(\log(\lambda))$  or  $O(1)$ . Apart from the homomorphic PRG evaluation, the online phase increases the multiplication depth by 2. Overall, the depth of homomorphic multiplications is  $d + 2$ , where either  $d = O(\log(\lambda))$  or  $d = O(1)$ . When a low-complexity PRG (e.g., [App12, AIK04, AIK08, AK19, CM01]) is used, the homomorphic PRG evaluation results in an amortized cost of  $O(1)$  homomorphic addition/multiplication operations per gate. For each gate, the processes of circuit garbling, circuit evaluation and consistency check also use  $O(1)$  homomorphic addition/multiplication operations. In all, the amortized computation cost per gate is dominated by the constant number of homomorphic additions and multiplications.

In the  $(\mathcal{F}_{2\text{PSWHE}}, \mathcal{F}_{\text{Rand}})$ -hybrid model, the rounds of communication are to transmit the bits  $\{\sigma_{i,A}\}_{i \in \mathcal{I}_A}$ ,  $\{\sigma_{i,B}\}_{i \in \mathcal{I}_B}$ ,  $\{d_i\}_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}}$ ,  $\{\tilde{d}_i\}_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}}$ ,  $\{\pi_i\}_{i \in \mathcal{O}}$  and string  $V_A$ . Under full-duplex networks (e.g., most wired communication) where communication in both directions can happen simultaneously, our protocol  $\Pi_{2\text{PC}}$  takes only four rounds, where  $\{\sigma_{i,A}\}_{i \in \mathcal{I}_A}$  and  $\{\sigma_{i,B}\}_{i \in \mathcal{I}_B}$  can be sent in parallel;  $\{\pi_i\}_{i \in \mathcal{O}}$  and  $V_A$  can be sent simultaneously. For half-duplex networks (e.g., most wireless communication), the protocol needs five rounds. It is well-known that  $\mathcal{F}_{\text{Rand}}$  can be instantiated by a two-round protocol (resp., a three-round protocol) in the random-oracle model for full-duplex networks (resp., half-duplex networks). We can also use the Fiat-Shamir heuristic to generate the random challenges  $\chi_i$  for  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  and  $\chi'_i$  for  $i \in \mathcal{O}$ . That is, both parties compute the challenges by hashing the transcript to this point. In this case, there are no rounds for producing the challenges. When  $\mathcal{F}_{2\text{PSWHE}}$  is instantiated by our protocol shown in Section 5.2, the rounds to securely realize  $\mathcal{F}_{2\text{PSWHE}}$  is  $O(1)$ . Overall, our 2PC protocol with active security has constant rounds.

**Optimizations.** We propose two optimizations to further improve the efficiency of protocol  $\Pi_{2\text{PC}}$  (Figures 4 and 5). One is to reuse the ciphertexts  $\llbracket s_A \rrbracket$  and  $\llbracket s_B \rrbracket$  between two parties; the other is to reduce the total communication cost to  $1 + o(1)$  bits per gate.

When using a low-depth weak pseudorandom function (wPRF) such as [BIP<sup>+</sup>18, DGH<sup>+</sup>21, APRR24] instead of PRG, both parties can reuse  $\llbracket s_A \rrbracket$  and  $\llbracket s_B \rrbracket$  for life. Specifically, for each 2PC execution,  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{Rand}}$  to generate a common randomness  $\text{nonce} \in \{0, 1\}^\lambda$ . Let  $\text{wPRF} : \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}$  be a low-depth wPRF and  $H : \{0, 1\}^{2\lambda} \rightarrow \{0, 1\}^\lambda$  be a random oracle. Then  $P_A$  can compute  $\hat{r}_i := \text{wPRF}(s_A, H(\text{nonce}, 0, i))$  and  $r_i := \text{wPRF}(s_A, H(\text{nonce}, 1, i))$ , while  $P_B$  computes  $\hat{t}_i := \text{wPRF}(s_B, H(\text{nonce}, 2, i))$  and  $t_i := \text{wPRF}(s_B, H(\text{nonce}, 3, i))$ . The corresponding ciphertexts can be computed homomorphically by both parties. In this case, except for the first setup execution and instantiating  $\mathcal{F}_{\text{Rand}}$ , the preprocessing phase has no communication, and only involves local computation.

The second optimization relies on the fact that both parties do not individually check the correctness of each gate. Instead, the circuit  $f$  is divided into smaller sequential subcircuits  $f = f_1 \circ f_2 \circ \dots \circ f_\ell$ , where

each subcircuit  $f_i$  has a multiplication depth of at most  $L$ . We only need to check if each  $f_i$  is computed correctly and whether the output of  $f_{i-1}$  match with the input of  $f_i$ , using the method implied in protocol  $\Pi_{2PC}$ . Specifically, for the output bits of each subcircuit  $f_i$ , we obtain the corresponding ciphertexts by running the protocol  $\Pi_{2PC}$ , which is necessary to control the multiplication depth of the SWHE scheme. Note that the ciphertexts on the output bits of  $f_{i-1}$  are directly used as the input ciphertexts for  $f_i$ , and thus it is natural that the output of  $f_{i-1}$  match with the input of  $f_i$ . Then, if the SWHE scheme additionally supports  $L$  multiplication depth, we are able to compute the ciphertexts on the output bits of each  $f_i$  via homomorphic evaluation, which is used to check the correctness of circuit evaluation. Finally, both parties can run the procedure of consistency check in protocol  $\Pi_{2PC}$  to verify whether the plaintexts encrypted in two kinds of ciphertexts (i.e., one kind of ciphertexts is generated by  $\Pi_{2PC}$  and the other is produced by homomorphic evaluation) are consistent.

Through this optimization, we have the following results:

1. For layered circuits where the gates on the  $i$ -th layer take only the output bits from that on the  $(i-1)$ -th layer as input, the communication per gate is optimized to  $1 + O(1/L)$  bits, where the total number of output wires for all  $f_i$  is  $O(C/L)$ . When  $L$  is set as  $\omega(1)$ , the communication is  $1 + o(1)$  bits per gate.
2. For arbitrary circuits, using the low-diameter decomposition [Bar96], a circuit with  $C$  gates can be divided into a series of subcircuits (i.e.,  $f_1, f_2, \dots, f_\ell$ ) with each having the depth of at most  $L$ , where the total number of output wires for all  $f_i$  is bounded by  $O(C \log C/L)$ . Consequently, the communication per gate is  $1 + O(\log C/L)$  bits. By setting  $L = \omega(\log C) = \omega(\log \lambda)$ , the communication for checking becomes  $o(1)$  bits per gate. As a result, the communication is reduced to  $1 + o(1)$  bits per gate.

## 4.2 Correctness Analysis

We prove the correctness of protocol  $\Pi_{2PC}$  (Figures 4 and 5) as below. Following the correctness analysis of BitGC [LWYY25], we replace  $\text{PRF}(\mathbf{k}, i)$  with  $R_i \stackrel{\$}{\leftarrow} \mathcal{R}_p$ , for each wire  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ . If the probability that the correctness does not hold for  $\text{PRF}(\mathbf{k}, i)$  but it holds for  $R_i$  is not negligible, then we can break the pseudorandomness of PRF with noticeable probability. Below, we show the correctness of our protocol for the case that  $\text{PRF}(\mathbf{k}, i)$  is replaced with  $R_i \stackrel{\$}{\leftarrow} \mathcal{R}_p$ , where  $W_{i,A} \in \mathcal{R}_p$  is uniform. In this case, the properties of both linear distributed decryption and correlated-key distributed decryption hold.

**The correctness of input processing.** We show that for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B$ ,  $P_A$  obtains a 0-label  $W_{i,A}$ , and  $P_B$  gets a label  $W_{i,B} = W_{i,A} + (-1)^{\pi_i} \cdot v_i \cdot \Delta$ , where  $\pi_i = \text{LSB}(W_{i,A})$  and  $v_i$  denotes the actual bit. Through steps 5 and 6,  $P_A$  and  $P_B$  obtain the ciphertext  $\llbracket v_i \rrbracket$  for  $i \in \mathcal{I}_A \cup \mathcal{I}_B$ . For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B$ , using the definitions of  $\widetilde{W}_{i,A}$  and  $W_{i,A}$ , we have

$$\begin{aligned}\widetilde{W}_{i,A} &:= \text{Dec}(\Delta_A, \llbracket (-1)^{r_i} \cdot v_i \rrbracket) + R_i, \\ W_{i,A} &:= \text{Dec}(\Delta_A, \llbracket (-1)^{\pi_i} \cdot v_i \rrbracket) + R_i,\end{aligned}$$

where  $R_i \stackrel{\$}{\leftarrow} \mathcal{R}_p$  and  $v_i = v_i^*$ . From  $\llbracket (-1)^{r_i} \cdot v_i \rrbracket = (-1)^{d_i} \cdot \llbracket (-1)^{\pi_i} \cdot v_i \rrbracket$ , we have  $\widetilde{W}_{i,A} = (-1)^{d_i} \cdot W_{i,A}$  from the property of linear distributed decryption. Using the properties  $\text{LSB}(-X) = \text{LSB}(X)$  for any  $X \in \mathcal{R}_p$  and an even  $p$ , it follows that  $\text{LSB}(\widetilde{W}_{i,A}) = \text{LSB}(W_{i,A})$ . Hence,  $\pi_i = \text{LSB}(W_{i,A})$  based on the fact that  $\pi_i = \text{LSB}(\widetilde{W}_{i,A})$ . From the property of linear distributed decryption, we have

$$W_{i,B} = W_{i,A} + \text{Dec}(\Delta, \llbracket (-1)^{\pi_i} \cdot v_i \rrbracket) = W_{i,A} + (-1)^{\pi_i} \cdot v_i \cdot \Delta.$$

**The correctness of circuit garbling and evaluation.** For each wire  $i$ , we denote  $v_i$  as the actual bit using the real inputs, and use  $u_i$  to denote the actual bits when all inputs are 0. Then, we prove the following invariant by induction:

- For each wire  $i$ , given that  $P_A$  holds  $(W_{i,A}, \pi_i)$ , and  $P_B$  holds  $W_{i,B}$ , we have  $\pi_i = \text{LSB}(W_i^0)$ , and the 0-label  $W_i^0$  and the labels  $W_{i,A}, W_{i,B}$  satisfy

$$W_i^0 \stackrel{\text{def}}{=} W_{i,A} - (-1)^{\pi_i} \cdot u_i \cdot \Delta \text{ and } W_{i,B} = W_i^0 + (-1)^{\pi_i} \cdot v_i \cdot \Delta.$$

According to the invariant, we straightforwardly have that  $\text{LSB}(W_{i,A}) = \pi_i \oplus u_i$  and  $W_{i,B} = W_{i,A} + (-1)^{\pi_i} \cdot (v_i - u_i) \cdot \Delta$ .

First, for each circuit-input wire  $i \in \mathcal{I}_A \cup \mathcal{I}_B$ , the above invariant trivially holds where  $u_i = 0$ . For each gate  $(a, b, c, g)$  in topological order, assuming that the invariant holds for input wires  $a$  and  $b$ , we prove that it also holds for output wire  $c$ . According to the definitions of  $\widetilde{W}_{c,A}$  and  $W_{c,A}$ , we have

$$\begin{aligned} \widetilde{W}_{c,A} &:= \text{Eval}(W_{a,A}, W_{b,A}, \widetilde{\tau}_1, \widetilde{\tau}_2, \widetilde{\tau}_3) + R_c, \\ W_{c,A} &:= \text{Eval}(W_{a,A}, W_{b,A}, \tau_1, \tau_2, \tau_3) + R_c, \end{aligned}$$

where  $\tau_1 = (-1)^{d_c} \cdot \widetilde{\tau}_1$ ,  $\tau_2 = (-1)^{d_c} \cdot \widetilde{\tau}_2$  and  $\tau_3 = (-1)^{d_c} \cdot \widetilde{\tau}_3$ . Therefore, we have  $\widetilde{W}_{c,A} = (-1)^{d_c} \cdot W_{c,A}$  using the property of linear distributed decryption, and thus  $\text{LSB}(\widetilde{W}_{c,A}) = \text{LSB}(W_{c,A})$  based on the fact that  $\text{LSB}(-X) = \text{LSB}(X)$  for any  $X \in \mathcal{R}_p$  and an even  $p$ . According to the induction assumption, we have  $\alpha = \text{LSB}(W_{a,A}) = \pi_a \oplus u_a$  and  $\beta = \text{LSB}(W_{b,A}) = \pi_b \oplus u_b$ . Then we obtain  $\pi_c = \text{LSB}(\widetilde{W}_{c,A}) \oplus g(\pi_a \oplus \alpha, \pi_b \oplus \beta) = \text{LSB}(\widetilde{W}_{c,A}) \oplus g(u_a, u_b) = \text{LSB}(\widetilde{W}_{c,A}) \oplus u_c$ . Therefore, we let  $W_c^0 \stackrel{\text{def}}{=} W_{c,A} - (-1)^{\pi_c} \cdot u_c \cdot \Delta$ ; we have  $\pi_c = \text{LSB}(\widetilde{W}_{c,A}) \oplus u_c = \text{LSB}(W_{c,A}) \oplus u_c = \text{LSB}(W_c^0)$ .

Then, both parties compute  $\widetilde{\tau}_1, \widetilde{\tau}_2$ , and  $\widetilde{\tau}_3$  as follows:

$$\begin{aligned} \widetilde{\tau}_1 &:= \llbracket (-1)^{r_c} \cdot z_{1,0} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket, \\ \widetilde{\tau}_2 &:= \llbracket (-1)^{r_c} \cdot z_{0,1} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket, \\ \widetilde{\tau}_3 &:= \llbracket (-1)^{r_c} \cdot z_{0,0} \rrbracket + \llbracket (-1)^{r_c} \cdot z_{1,1} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{1,0} \rrbracket - \llbracket (-1)^{r_c} \cdot z_{0,1} \rrbracket, \end{aligned}$$

and set  $\tau_i = (-1)^{d_c} \cdot \widetilde{\tau}_i$  for  $i \in [3]$ , where  $z_{i,j} = g(\pi_a \oplus i, \pi_b \oplus j) \in \{0, 1\}$  for  $i, j \in \{0, 1\}$ . Thus, we have

$$\begin{aligned} \text{Dec}(\Delta, \tau_1) &= (-1)^{d_c} \cdot \text{Dec}(\Delta, \widetilde{\tau}_1) = (-1)^{\pi_c} \cdot (z_{1,0} - z_{0,0}) \cdot \Delta, \\ \text{Dec}(\Delta, \tau_2) &= (-1)^{d_c} \cdot \text{Dec}(\Delta, \widetilde{\tau}_2) = (-1)^{\pi_c} \cdot (z_{0,1} - z_{0,0}) \cdot \Delta, \\ \text{Dec}(\Delta, \tau_3) &= (-1)^{d_c} \cdot \text{Dec}(\Delta, \widetilde{\tau}_3) = (-1)^{\pi_c} \cdot (z_{0,0} + z_{1,1} - z_{0,1} - z_{1,0}) \cdot \Delta. \end{aligned}$$

Let  $W_a^{\pi_a} \stackrel{\text{def}}{=} W_a^0 - \pi_a \cdot \Delta$  and  $W_b^{\pi_b} \stackrel{\text{def}}{=} W_b^0 - \pi_b \cdot \Delta$ , where  $\text{LSB}(W_a^{\pi_a}) = 0$  and  $\text{LSB}(W_b^{\pi_b}) = 0$ . For each  $i \in \{a, b\}$ , according to the definition of  $W_{i,A}$ , we have  $W_{i,A} = W_i^0 + (-1)^{\pi_i} \cdot u_i \cdot \Delta = W_i^{\pi_i} + (\pi_i + (-1)^{\pi_i} \cdot u_i) \cdot \Delta = W_i^{\pi_i} + (u_i \oplus \pi_i) \cdot \Delta$ . According to the induction assumption, we have that  $W_{i,B} = W_i^0 + (-1)^{\pi_i} \cdot v_i \cdot \Delta$  for each  $i \in \{a, b\}$ . Thus, for each  $i \in \{a, b\}$ ,  $W_{i,B} = W_i^{\pi_i} + (v_i \oplus \pi_i) \cdot \Delta$  via a similar analysis. Based on the property of Eval shown in Section 3.3, we have

$$\begin{aligned} &\text{Eval}(W_{a,A}, W_{b,A}, \tau_1, \tau_2, \tau_3) - \text{Eval}(W_a^{\pi_a}, W_b^{\pi_b}, \tau_1, \tau_2, \tau_3) \\ &= (u_a \oplus \pi_a) \cdot \text{Dec}(\Delta, \tau_1) + (u_b \oplus \pi_b) \cdot \text{Dec}(\Delta, \tau_2) + (u_a \oplus \pi_a) \cdot (u_b \oplus \pi_b) \cdot \text{Dec}(\Delta, \tau_3) \\ &= (-1)^{\pi_c} \cdot ((u_a \oplus \pi_a) \cdot (z_{1,0} - z_{0,0}) + (u_b \oplus \pi_b) \cdot (z_{0,1} - z_{0,0}) \\ &\quad + (u_a \oplus \pi_a)(u_b \oplus \pi_b) \cdot (z_{0,0} + z_{1,1} - z_{0,1} - z_{1,0})) \cdot \Delta \\ &= (-1)^{\pi_c} \cdot (z_{u_a \oplus \pi_a, u_b \oplus \pi_b} - z_{0,0}) \cdot \Delta \\ &= (-1)^{\pi_c} \cdot (g(u_a, u_b) - z_{0,0}) \cdot \Delta = (-1)^{\pi_c} \cdot (u_c - z_{0,0}) \cdot \Delta. \end{aligned}$$

Similarly, we have

$$\text{Eval}(W_{a,B}, W_{b,B}, \tau_1, \tau_2, \tau_3) = \text{Eval}(W_a^{\pi_a}, W_b^{\pi_b}, \tau_1, \tau_2, \tau_3) + (-1)^{\pi_c} \cdot (v_c - z_{0,0}) \cdot \Delta.$$

Thus, we have

$$\begin{aligned} W_{c,B} &= \text{Eval}(W_{a,B}, W_{b,B}, \tau_1, \tau_2, \tau_3) + R_c \\ &= \text{Eval}(W_a^{\pi_a}, W_b^{\pi_b}, \tau_1, \tau_2, \tau_3) + (-1)^{\pi_c} \cdot (v_c - z_{0,0}) \cdot \Delta + R_c \\ &= \text{Eval}(W_{a,A}, W_{b,A}, \tau_1, \tau_2, \tau_3) + (-1)^{\pi_c} \cdot (v_c - u_c) \cdot \Delta + R_c \\ &= W_{c,A} + (-1)^{\pi_c} \cdot (v_c - u_c) \cdot \Delta. \end{aligned}$$

Let  $W_c^0 \stackrel{\text{def}}{=} W_{c,A} - (-1)^{\pi_c} \cdot u_c \cdot \Delta$ . We obtain  $W_{c,B} = W_c^0 + (-1)^{\pi_c} \cdot v_c \cdot \Delta$ . From the above analysis, for each  $i \in \mathcal{O}$ , we have the following:

$$\pi_i = \text{LSB}(W_{i,A}) \oplus u_i \quad \text{and} \quad W_{i,B} = W_{i,A} + (-1)^{\pi_i} \cdot (v_i - u_i) \cdot \Delta.$$

After receiving  $\pi_i$  from  $\mathcal{P}_A$ ,  $\mathcal{P}_B$  computes an output bit  $\text{LSB}(W_{i,B}) \oplus \pi_i$ . Therefore, the output bit is equal to  $\text{LSB}(W_{i,A}) \oplus u_i \oplus v_i \oplus \pi_i = v_i$ , based on the fact  $\text{LSB}(W_{i,B}) = \text{LSB}(W_{i,A}) \oplus u_i \oplus v_i$ .

**The correctness of consistency check.** According to the above correctness analysis, we have  $\text{LSB}(W_{i,A}) = \pi_i \oplus u_i$  and  $\text{LSB}(W_{i,B}) = \text{LSB}(W_{i,A}) \oplus u_i \oplus v_i$  for each wire  $i$ . Therefore, we obtain  $\text{LSB}(W_{i,B}) = \pi_i \oplus v_i$ , where  $v_i$  is the actual bit for the real inputs. From the definition of  $\tilde{d}_i$ , we have  $\tilde{d}_i = t_i \oplus \pi_i \oplus v_i$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ . Based on the correctness of SWHE, we obtain that  $\llbracket v_i \rrbracket = \llbracket \pi_i \rrbracket \oplus \llbracket t_i \rrbracket \oplus \tilde{d}_i$  encrypts the actual bit  $v_i$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ . For each gate  $(a, b, c, g)$ , it is easy to see that  $\llbracket v_c^* \rrbracket = g(\llbracket v_a \rrbracket, \llbracket v_b \rrbracket)$  encrypts the actual bit  $v_c^* = v_c$ . In the remaining part, by working through the protocol, it is natural to verify that the consistency check is correct, which concludes the analysis.

### 4.3 Proof of Security

Before diving into the details of security proof for protocol  $\Pi_{2PC}$  (Figures 4 and 5), we give an overview of the proof. There is no interaction in the hybrid model for the preprocessing phase, and thus the simulation is trivial by emulating  $\mathcal{F}_{2PSWHE}$ . Below, we first consider the case of malicious  $\mathcal{P}_A$ , and then focus on the case of malicious  $\mathcal{P}_B$ .

**Malicious  $\mathcal{P}_A$ .** We first show the construction of a probabilistic polynomial time (PPT) simulator  $\mathcal{S}$ . Specifically,  $\mathcal{S}$  can extract the  $\mathcal{P}_A$ 's input by computing  $x_i := \sigma_{i,A} \oplus \hat{r}_i$  for each  $i \in \mathcal{I}_A$ , as it knows all  $\mathcal{P}_A$ 's secrets by emulating  $\mathcal{F}_{2PSWHE}$ . Then,  $\mathcal{S}$  sends a random bit  $\sigma_{i,B}$  to the PPT adversary  $\mathcal{A}$  for each  $i \in \mathcal{I}_B$ . For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ , after receiving  $d_i \in \{0, 1\}$  from  $\mathcal{A}$ ,  $\mathcal{S}$  is able to compute a correct bit  $d_i^*$  by simulating the  $\mathcal{P}_A$ 's computation honestly with the  $\mathcal{P}_A$ 's secrets. On behalf of honest  $\mathcal{P}_B$ ,  $\mathcal{S}$  can send a uniform bit  $\tilde{d}_i$  to  $\mathcal{A}$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ . For each  $i \in \mathcal{O}$ , after receiving  $\pi_i \in \{0, 1\}$  from  $\mathcal{A}$ ,  $\mathcal{S}$  can compute a correct permutation bit  $\pi_i^* \in \{0, 1\}$  by itself. Finally,  $\mathcal{S}$  receives  $V_A$  from  $\mathcal{A}$ , and aborts if there exists some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  such that  $d_i \neq d_i^*$  or some  $i \in \mathcal{O}$  such that  $\pi_i \neq \pi_i^*$ . To handle the case that  $\mathcal{A}$  sends a ‘‘garbage’’ string  $V_A$ ,  $\mathcal{S}$  also checks if  $V_A = H(\text{Dec}(\Delta_A, \tau), \tau)$ .

Obviously,  $x_i$  is extracted perfectly. The difference between  $\{\sigma_{i,B}\}_{i \in \mathcal{I}_B} \cup \{\tilde{d}_i\}_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}}$  sent in the real protocol execution and that simulated by  $\mathcal{S}$  can be bounded by the CPA security of the SWHE scheme and pseudorandomness of PRG output. In particular, through the CPA security of SWHE, we can replace  $\llbracket s_B \rrbracket$  with  $\llbracket 0 \rrbracket$ . Then, based on the pseudorandomness of PRG, we can replace all  $\hat{t}_i, t_i$  with uniform bits. In the following, we analyze the difference of checking  $V_A$  between the real-world execution and ideal-world execution. Let  $e_i \stackrel{\text{def}}{=} v_i^* - v_i$  for  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  and  $e'_i \stackrel{\text{def}}{=} \hat{\pi}_i - \pi_i$  for  $i \in \mathcal{O}$ , where  $\hat{\pi}_i \stackrel{\text{def}}{=} r_i \oplus d_i$ . If  $d_i \neq d_i^*$

for some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ , then at least one  $e_i \neq 0$ . Otherwise, if  $\pi_i \neq \pi_i^*$  for some  $i \in \mathcal{O}$ , at least one  $e'_i \neq 0$ , and  $\pi_i \neq \hat{\pi}_i$  as  $\hat{\pi}_i = \pi_i^*$  in this case. In the real protocol execution, if  $V_A = V_B$ , we have

$$\sum_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}} \chi_i \cdot e_i + \sum_{i \in \mathcal{O}} \chi'_i \cdot e'_i = 0,$$

except with probability  $\text{negl}(\lambda)$ . Since the challenges  $\chi_i, \chi'_i \in \mathbb{Z}_p$  are sampled uniformly after  $e_i, e'_i$  have been determined, both  $e_i = 0$  and  $e'_i = 0$ , except with probability  $1/p$ . Note that  $e_i, e'_i \in \{-1, 0, 1\}$ . Therefore, with overwhelming probability, we have that all  $d_i$  and  $\pi_i$  sent by  $\mathcal{A}$  are correct, if the protocol execution does not abort. According to the correctness analysis, honest  $P_B$  obtains the correct output in the real world, under the pseudorandomness of PRF outputs.

**Malicious  $P_B$ .** As such, we first show the simulation of  $\mathcal{S}$ . Specifically,  $\mathcal{S}$  sends a uniform bit  $\sigma_{i,A}$  to  $\mathcal{A}$  for each  $i \in \mathcal{I}_A$ . For each  $i \in \mathcal{I}_B$ , after receiving  $\sigma_{i,B}$  from  $\mathcal{A}$ ,  $\mathcal{S}$  can extract the input bit  $y_i := \sigma_{i,B} \oplus \hat{t}_i$  for each  $i \in \mathcal{I}_B$ . Then,  $\mathcal{S}$  sends the  $P_B$ 's input to  $\mathcal{F}_{2PC}$ , and obtains the output bits  $z_i^*$  for all  $i \in \mathcal{O}$ . Then, for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ ,  $\mathcal{S}$  sends a random bit  $d_i$  to  $\mathcal{A}$ . For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ , after receiving  $\tilde{d}_i \in \{0, 1\}$  from  $\mathcal{A}$ ,  $\mathcal{S}$  computes a correct bit  $\tilde{d}_i^*$  following  $P_B$ 's computation honestly, where  $\mathcal{S}$  knows all  $P_B$ 's secrets (e.g.,  $\Delta_B, s_B$ ) by emulating  $\mathcal{F}_{2PSWHE}$ . For each  $i \in \mathcal{O}$ ,  $\mathcal{S}$  computes  $\pi_i := z_i^* \oplus \text{LSB}(W_{i,B})$  and sends it to  $\mathcal{A}$ , where  $W_{i,B}$  is computed following the protocol specification. If there exists some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  such that  $\tilde{d}_i \neq \tilde{d}_i^*$ ,  $\mathcal{S}$  samples a random  $V_A$  and sends it to  $\mathcal{A}$ . Otherwise,  $\mathcal{S}$  computes  $V_B$  following the protocol description and sends it to  $\mathcal{A}$ .

The extraction of  $y_i$  is also perfect. The simulation of all  $\sigma_{i,A}$  and  $d_i$  is indistinguishable from the bits sent in the real protocol execution, using a similar reduction based on that SWHE is CPA secure and the output of PRG is pseudorandom. In the remaining part, we analyze the difference between  $V_A$  sent in the real-world execution and that simulated by  $\mathcal{S}$ . If there exists some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  such that  $\tilde{d}_i \neq \tilde{d}_i^*$ , then there is at least one  $v_i = \pi_i \oplus t_i \oplus \tilde{d}_i$  that is not equal to  $v_i^*$ . In this case,  $\tau$  encrypts one non-zero element  $E$  and  $V_A \neq V_B$  in the real-world execution. Otherwise, using a similar analysis, we have that  $e_i \stackrel{\text{def}}{=} v_i^* - v_i = 0$  with probability  $1 - \text{negl}(\lambda)$ . Therefore,  $V_A$  is computationally indistinguishable from a uniform string in the random-oracle model, as  $\text{Dec}(\Delta_A, \tau)$  is kept unknown against  $\mathcal{A}$  under the assumption that SWHE is CPA secure for any pair  $(\text{keypar}', \Delta) \leftarrow \text{KeyGen}(\text{params}, \Delta_A, \Delta_B)$ . In particular, if  $\mathcal{A}$  makes a query  $(\text{Dec}(\Delta_A, \tau), \tau)$  to random oracle  $H$ , then the reduction can extract the query and compute  $\text{Dec}(\Delta, \tau) = \text{Dec}(\Delta_B, \tau) - \text{Dec}(\Delta_A, \tau)$ , where  $\Delta_B - \Delta_A = \Delta$ . Thus, the reduction obtains  $\text{Dec}(\Delta, \tau) = E \cdot \Delta$ , and recovers secret key  $\Delta$  as  $E \neq 0$ . If  $\tilde{d}_i = \tilde{d}_i^*$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ , we can prove that  $v_i = v_i^*$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  is correct, according to the correctness analysis under the pseudorandomness of PRF outputs. Therefore, we have  $V_A = V_B$ , which completes the overview of the proof.

**Theorem 1.** *Let  $f$  be a two-party functionality  $\{0, 1\}^{|\mathcal{I}_A|} \times \{0, 1\}^{|\mathcal{I}_B|} \rightarrow \{0, 1\}^{|\mathcal{O}|}$  with  $|\mathcal{I}_A| = |\mathcal{I}_B|$ . Let PRG be a pseudorandom generator, PRF be a pseudorandom function, and  $H$  be a random oracle. Assume that the SWHE scheme is CPA secure under any pair  $(\text{keypar}', \Delta) \leftarrow \text{KeyGen}(\text{params}, \Delta_A, \Delta_B)$ , if at least one party is honest. Then protocol  $\Pi_{2PC}$  (Figures 4 and 5) securely computes  $f$  with statistical error  $1/p$  in the presence of a static, malicious adversary in the  $(\mathcal{F}_{2PSWHE}, \mathcal{F}_{\text{Rand}})$ -hybrid model.*

The detailed proof of Theorem 1 is given in Appendix B.1. Note that  $p \geq 2^\rho$  for a statistical security parameter  $\rho$ . In the formal proof shown in Appendix B.1, we use a sequence of hybrids to prove the indistinguishability between the real-world execution and ideal-world execution. We first change the manner of consistency check for the case of malicious  $P_A$ , or the computation manner of  $V_A$  for the case of malicious  $P_B$ . In both cases, the checking procedure does not involve the honest party's secret key. In the subsequent hybrid, we can construct a reduction against the CPA security of the SWHE scheme to bound the difference between either  $\llbracket s_A \rrbracket$  or  $\llbracket s_B \rrbracket$  and  $\llbracket 0 \rrbracket$ , since the checking procedure does not include the secret key.



## 5 Instantiation for Realizing Ideal Functionality $\mathcal{F}_{2\text{PSWHE}}$

Our main protocol shown in Section 4 calls an ideal functionality  $\mathcal{F}_{2\text{PSWHE}}$  to encrypt short seeds. Moreover,  $\mathcal{F}_{2\text{PSWHE}}$  can be called only once in the setup phase. Thus, even if we use a generic 2PC with active security to instantiate  $\mathcal{F}_{2\text{PSWHE}}$ , the cost may be fairly small compared to the overall cost, and the rounds only depend on key generation and encryption of SWHE, and are independent of the circuit depth (i.e., the round complexity keeps  $O(1)$ ). In this section, we present a practical instantiation to achieve better efficiency. First of all, we recall the RLWE-based instantiation [LWYY25] for SWHE with distributed decryption in Section 5.1. Then, we describe a two-party actively secure protocol to realize  $\mathcal{F}_{2\text{PSWHE}}$  securely.

### 5.1 RLWE-Based SWHE Scheme with Distributed Decryption

Below, we recall the RLWE-based SWHE scheme with distributed decryption, which is referred to as an extended GSW (eGSW) scheme in [LWYY25]. It builds upon the GSW HE scheme [GSW13], and additionally supports the distributed decryption property. We first summarize some key functions used in the SWHE scheme with distributed decryption [LWYY25]: BitDecomp, BitDecomp<sup>-1</sup>, Flatten, and Powersof2.

- **BitDecomp:** For any  $a \in \mathcal{R}_q$ , BitDecomp( $a$ ) produces a vector  $(a_0, \dots, a_{\ell-1})$  of length  $\ell = \lfloor \log(q) \rfloor + 1$ , where each  $a_i \in \mathcal{R}_2$  represents the  $i$ -th bit component of  $a$ , with  $\mathcal{R}_2 \stackrel{\text{def}}{=} \mathbb{Z}_2[X]/(X^n + 1)$ .
- **BitDecomp<sup>-1</sup>:** For  $\mathbf{a} = (a_0, \dots, a_{\ell-1}) \in \mathcal{R}_q^\ell$ , BitDecomp<sup>-1</sup>( $\mathbf{a}$ )  $\stackrel{\text{def}}{=} \sum_{i=0}^{\ell-1} (2^i \cdot a_i)$ .
- **Flatten:** For each  $\mathbf{a} = (a_0, \dots, a_{\ell-1}) \in \mathcal{R}_q^\ell$ , to obtain a bit vector, Flatten( $\mathbf{a}$ )  $\stackrel{\text{def}}{=} \text{BitDecomp}(\text{BitDecomp}^{-1}(\mathbf{a}))$ .
- **Powersof2:** For  $b \in \mathcal{R}_q$ , Powersof2( $b$ )  $\stackrel{\text{def}}{=}} (2^0 \cdot b, 2^1 \cdot b, \dots, 2^{\ell-1} \cdot b) \in \mathcal{R}_q^\ell$ .

For  $\mathbf{a} \in \mathcal{R}_q^\ell$  and  $b \in \mathcal{R}_q$ , the following property holds:

$$\langle \mathbf{a}, \text{Powersof2}(b) \rangle = \text{BitDecomp}^{-1}(\mathbf{a}) \cdot b = \langle \text{Flatten}(\mathbf{a}), \text{Powersof2}(b) \rangle.$$

The operations mentioned as above can be extended to matrices by applying them to each entry in a matrix. In the following, the eGSW SWHE scheme with distributed decryption [LWYY25] is described as follows.

**Setup.** Setup( $1^\lambda, 1^L$ ) outputs a set of parameters  $\text{params} = (n, p, q, \eta, \ell, N)$ , where  $p = \lambda^{\omega(1)}$  is an even,  $p \mid q, q/p^2 = \lambda^{\omega(1)} \cdot (N+1)^L \cdot n^L$ ,  $\eta = \text{poly}(\lambda)$ ,  $\ell = \lfloor \log(q) \rfloor + 1$ , and  $N = 2\ell$ .

**Key generation.** KeyGen( $\text{params}$ ) samples  $\text{sk}, \widehat{\text{sk}} \leftarrow \mathcal{D}_\eta$  with  $\text{sk}[1] = \widehat{\text{sk}}[1] = 1$ , and then sets  $\text{sk} = \text{sk}_B - \text{sk}_A$  and  $\widehat{\text{sk}} = \widehat{\text{sk}}_B - \widehat{\text{sk}}_A$ . Then, KeyGen( $\text{params}$ ) outputs a set of key-related parameters  $\text{keypar}$ , including ciphertexts  $\tau_{\text{sk} \rightarrow \widehat{\text{sk}}}$ ,  $\tau_{\widehat{\text{sk}} \rightarrow \text{sk}}$ , and  $\tau_{\text{gsw} \rightarrow \text{egsw}}$ , constructed as follows:

- **The construction of  $\tau_{\text{sk} \rightarrow \widehat{\text{sk}}}$  and  $\tau_{\widehat{\text{sk}} \rightarrow \text{sk}}$ .** Sample  $a, \widehat{a} \xleftarrow{\$} \mathcal{R}_q$  and  $e, \widehat{e} \leftarrow \mathcal{D}_\eta$ , and then compute two key-switching ciphertexts  $\tau_{\text{sk} \rightarrow \widehat{\text{sk}}}$  and  $\tau_{\widehat{\text{sk}} \rightarrow \text{sk}}$  as follows:

$$\begin{aligned} \tau_{\text{sk} \rightarrow \widehat{\text{sk}}} &= (a, b = a \cdot \text{sk} + e + (q/p) \cdot \widehat{\text{sk}}), \\ \tau_{\widehat{\text{sk}} \rightarrow \text{sk}} &= (\widehat{a}, \widehat{b} = \widehat{a} \cdot \widehat{\text{sk}} + \widehat{e} + (q/p) \cdot \text{sk}). \end{aligned}$$

- **The construction of  $\tau_{\text{gsw} \rightarrow \text{egsw}}$ .** Sample  $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^N$  and  $e^1, e^2, e^3 \leftarrow \mathcal{D}_\eta^N$ , and then compute a GSW-to-eGSW ciphertext  $\tau_{\text{gsw} \rightarrow \text{egsw}}$  as follows:

$$\tau_{\text{gsw} \rightarrow \text{egsw}} = (\mathbf{a}, \mathbf{b}^1 = \mathbf{a} \cdot \text{sk} + e^1, \mathbf{b}^2 = \mathbf{a} \cdot \widehat{\text{sk}} + e^2, \mathbf{b}^3 = \mathbf{b}^1 \cdot \widehat{\text{sk}} + e^3 + \mathbf{v} \cdot \widehat{\text{sk}}),$$

where  $\mathbf{v} \stackrel{\text{def}}{=} \text{Powersof2}([- \text{sk} \quad 1])^\top$ .



Macro EncMsg( $\mathbf{a}, \ell, \text{sk}_A, \text{sk}_B, \mathbf{m}_A, \mathbf{m}_B$ )

**Inputs:**  $P_A$  and  $P_B$  have the public inputs  $\mathbf{a} \in \mathcal{R}_q^\ell$  and  $\ell \in \mathbb{N}$ .  $P_A$  holds secret inputs  $\text{sk}_A \in \mathcal{D}_\eta$ ,  $\mathbf{m}_A \in \mathcal{R}^\ell$ , and  $P_B$  holds secret inputs  $\text{sk}_B \in \mathcal{D}_\eta$ ,  $\mathbf{m}_B \in \mathcal{R}^\ell$ , where  $\text{sk}_A, \text{sk}_B$  have been committed by functionality  $\mathcal{F}_{\text{CPC}}$ . Here, either  $\mathbf{m}_A, \mathbf{m}_B$  are also committed by  $\mathcal{F}_{\text{CPC}}$ , or both  $\mathbf{m}_A, \mathbf{m}_B$  are equal to zero.

**Encryption:**  $P_A$  and  $P_B$  jointly generate a ciphertext as follows.

1.  $P_A$  samples  $e_A \leftarrow \mathcal{D}_\eta^\ell$ , computes  $\mathbf{b}_A := \mathbf{a} \cdot \text{sk}_A + e_A + (q/p) \cdot \mathbf{m}_A \in \mathcal{R}_q^\ell$ , and sends  $\mathbf{b}_A$  to  $P_B$ . Both parties call the Prove command of  $\mathcal{F}_{\text{CPC}}$  to convince  $P_B$  that the following relationship holds:

$$\{(\text{sk}_A, e_A, \mathbf{m}_A) \mid \mathbf{b}_A = \mathbf{a} \cdot \text{sk}_A + e_A + (q/p) \cdot \mathbf{m}_A \text{ and } \|e_A\|_\infty \leq \eta\}.$$

2.  $P_B$  samples  $e_B \leftarrow \mathcal{D}_\eta^\ell$ , computes  $\mathbf{b}_B := \mathbf{a} \cdot \text{sk}_B + e_B + (q/p) \cdot \mathbf{m}_B \in \mathcal{R}_q^\ell$ , and sends  $\mathbf{b}_B$  to  $P_A$ . Both parties call the Prove command of  $\mathcal{F}_{\text{CPC}}$  to convince  $P_A$

$$\{(\text{sk}_B, e_B, \mathbf{m}_B) \mid \mathbf{b}_B = \mathbf{a} \cdot \text{sk}_B + e_B + (q/p) \cdot \mathbf{m}_B \text{ and } \|e_B\|_\infty \leq \eta\}.$$

3. Both parties locally compute  $\mathbf{b} := \mathbf{b}_B - \mathbf{b}_A \in \mathcal{R}_q^\ell$  with  $\mathbf{b} = \mathbf{a} \cdot \text{sk} + e + (q/p) \cdot \mathbf{m}$ , where  $\text{sk} \stackrel{\text{def}}{=} \text{sk}_B - \text{sk}_A \in \mathcal{D}_{2\eta}$ ,  $e \stackrel{\text{def}}{=} e_B - e_A \in \mathcal{D}_{2\eta}^\ell$  and  $\mathbf{m} \stackrel{\text{def}}{=} \mathbf{m}_B - \mathbf{m}_A \in \mathcal{R}_q^\ell$ .

Figure 6: Encryption macro in the  $\mathcal{F}_{\text{CPC}}$ -hybrid model, where the macro would be invoked by protocol  $\Pi_{2\text{PSWHE}}$  (Figures 7 and 8).

**Encryption.** For any message  $m \in \mathbb{Z}_p$ ,  $\text{Enc}(\text{sk}, m)$  works as follows: it samples  $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^N$  and  $e \leftarrow \mathcal{D}_\eta^N$ , computes  $\mathbf{b} := \mathbf{a} \cdot \text{sk} + e \in \mathcal{R}_q^N$  and outputs the ciphertext

$$\llbracket m \rrbracket := \text{Flatten} (m \cdot \mathbf{I}_N + \text{BitDecomp}([\mathbf{a} \ \mathbf{b}])) ,$$

where  $\mathbf{I}_N$  is the  $N \times N$  identity matrix.

**Decryption.** On input a key  $x \in \mathcal{R}$  (it is either the secret key  $\text{sk}$  or an additive share of  $\text{sk}$ ) and a ciphertext  $\tau \in \mathcal{R}_2^{N \times N}$ ,  $\text{Dec}(x, \tau)$  computes over  $\mathcal{R}_q$

$$(\alpha, \beta) := \text{BitDecomp}^{-1} (-\text{BitDecomp}([(q/p) \ 0]) \cdot \tau) ,$$

and then output  $\lfloor \text{LSB}(x) \cdot \beta - \alpha \cdot x \rfloor_p$ . Note that if  $x = \text{sk}$  with  $\text{LSB}(\text{sk}) = 1$ , then we have  $\beta - \alpha \cdot \text{sk} \approx (q/p) \cdot m \cdot \text{sk}$  and thus  $\lfloor \beta - \alpha \cdot \text{sk} \rfloor_p = m \cdot \text{sk}$ .

We postpone other algorithms of the above SWHE scheme to Appendix A, as these algorithms are unrelated for designing a protocol to instantiate  $\mathcal{F}_{2\text{PSWHE}}$ . Previous work [LWYY25] has proven that the above SWHE scheme is CPA secure under the RLWE assumption with key-dependent message (KDM) security. It also shows the distributed decryption property holds. As shown in [GSW13, LWYY25], the above scheme supports homomorphic addition and multiplication operations with multiplication depth being limited to at most  $L$ .

## 5.2 Our Two-Party SWHE Protocol with Active Security

We present an actively secure two-party protocol  $\Pi_{2\text{PSWHE}}$  in the  $(\mathcal{F}_{\text{CPC}}, \mathcal{F}_{\text{Rand}})$ -hybrid model, described in Figures 7 and 8. This protocol realizes distributed key generation and encryption of the above SWHE scheme, where the secret key  $\text{sk}$  is shared as  $\text{sk} = \text{sk}_B - \text{sk}_A$ , and another secret key  $\widehat{\text{sk}}$  is shared as  $\widehat{\text{sk}} = \widehat{\text{sk}}_B - \widehat{\text{sk}}_A$ . In

**Protocol  $\Pi_{2\text{PSWHE}}$  (Part 1)**

This protocol invokes a macro EncMsg defined in Figure 6. Suppose that  $P_A$  and  $P_B$  agree on a set of parameters  $\text{params} = (n, p, q, \eta, \ell, N, M) \leftarrow \text{Setup}(1^\lambda, 1^L)$  for RLWE-based SWHE with distributed decryption.

**Two-party key-generation protocol:**  $P_A$  and  $P_B$  jointly generate the secret keys and a set of key-related parameters  $\text{keypar}'$  as follows:

1.  $P_A$  samples  $\text{sk}_A, \widehat{\text{sk}}_A \leftarrow \mathcal{D}_\eta$  such that  $\text{sk}_A[1] = 0$  and  $\widehat{\text{sk}}_A[1] = 0$ , and calls the Commit command of  $\mathcal{F}_{\text{CPC}}$  to commit  $(\text{sk}_A, \widehat{\text{sk}}_A)$ . Then, both parties call the Prove command of  $\mathcal{F}_{\text{CPC}}$  to convince  $P_B$  that  $\|\text{sk}_A\|_\infty \leq \eta$ ,  $\|\widehat{\text{sk}}_A\|_\infty \leq \eta$ ,  $\text{sk}_A[1] = 0$  and  $\widehat{\text{sk}}_A[1] = 0$  hold.
2.  $P_B$  samples  $\text{sk}_B, \widehat{\text{sk}}_B \leftarrow \mathcal{D}_\eta$  such that  $\text{sk}_B[1] = 1$  and  $\widehat{\text{sk}}_B[1] = 1$ , and calls the Commit command of  $\mathcal{F}_{\text{CPC}}$  to commit  $(\text{sk}_B, \widehat{\text{sk}}_B)$ . Then, the parties call the Prove command of  $\mathcal{F}_{\text{CPC}}$  to convince  $P_A$  that  $\|\text{sk}_B\|_\infty \leq \eta$ ,  $\|\widehat{\text{sk}}_B\|_\infty \leq \eta$ ,  $\text{sk}_B[1] = 1$  and  $\widehat{\text{sk}}_B[1] = 1$  hold.
3.  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{Rand}}$  to sample random ring elements  $a, \widehat{a} \in \mathcal{R}_q$  along with uniform vectors  $\mathbf{a} \in \mathcal{R}_q^N$  and  $\widetilde{\mathbf{a}} \in \mathcal{R}_q^M$ .
4. Both parties execute  $\text{EncMsg}(a, 1, \text{sk}_A, \text{sk}_B, \widehat{\text{sk}}_A, \widehat{\text{sk}}_B)$  to obtain

$$\tau_{\text{sk} \rightarrow \widehat{\text{sk}}} \stackrel{\text{def}}{=} (a, b = a \cdot \text{sk} + e + (q/p) \cdot \widehat{\text{sk}})$$

for some  $e$  with  $\|e\|_\infty \leq 2\eta$ , where  $\text{sk} \stackrel{\text{def}}{=} \text{sk}_B - \text{sk}_A \in \mathcal{D}_{2\eta}$  with  $\text{sk}[1] = 1$  and  $\widehat{\text{sk}} \stackrel{\text{def}}{=} \widehat{\text{sk}}_B - \widehat{\text{sk}}_A \in \mathcal{D}_{2\eta}$  with  $\widehat{\text{sk}}[1] = 1$ .

5.  $P_A$  and  $P_B$  execute  $\text{EncMsg}(\widehat{a}, 1, \widehat{\text{sk}}_A, \widehat{\text{sk}}_B, \text{sk}_A, \text{sk}_B)$  to get

$$\tau_{\widehat{\text{sk}} \rightarrow \text{sk}} \stackrel{\text{def}}{=} (\widehat{a}, \widehat{b} = \widehat{a} \cdot \widehat{\text{sk}} + \widehat{e} + (q/p) \cdot \text{sk})$$

for some  $\widehat{e} \in \mathcal{D}_{2\eta}$ .

6.  $P_A$  and  $P_B$  execute  $\text{EncMsg}(\mathbf{a}, N, \text{sk}_A, \text{sk}_B, 0, 0)$  to generate  $\mathbf{b}^1 = \mathbf{a} \cdot \text{sk} + \mathbf{e}^1$  for some  $\mathbf{e}^1 \in \mathcal{D}_{2\eta}^N$ . In parallel, the parties run  $\text{EncMsg}(\mathbf{a}, N, \widehat{\text{sk}}_A, \widehat{\text{sk}}_B, 0, 0)$  to get  $\mathbf{b}^2 = \mathbf{a} \cdot \widehat{\text{sk}} + \mathbf{e}^2$  for some  $\mathbf{e}^2 \in \mathcal{D}_{2\eta}^N$ .
7. Both parties call  $\mathcal{F}_{\text{CPC}}$  to let  $P_A$  obtain  $x_A \in \mathcal{R}_q$  and  $P_B$  get  $x_B \in \mathcal{R}_q$ , such that  $x_B - x_A = \text{sk} \cdot \widehat{\text{sk}} \in \mathcal{R}_q$ . Then, using  $(x_A, \widehat{\text{sk}}_A)$  and  $(x_B, \widehat{\text{sk}}_B)$  respectively,  $P_A$  locally computes  $\mathbf{m}_A \in \mathcal{R}_q^N$ , and  $P_B$  locally computes  $\mathbf{m}_B \in \mathcal{R}_q^N$ , such that  $\mathbf{m}_B - \mathbf{m}_A = \mathbf{v} \cdot \widehat{\text{sk}} \in \mathcal{R}_q^N$ , where  $\mathbf{v} \stackrel{\text{def}}{=} \text{Powersof2}([- \text{sk} \ 1])^T$ .
8.  $P_A$  and  $P_B$  execute  $\text{EncMsg}(\mathbf{b}^1, N, \text{sk}_A, \text{sk}_B, \mathbf{m}_A, \mathbf{m}_B)$  to obtain  $\mathbf{b}^3 = \mathbf{b}^1 \cdot \widehat{\text{sk}} + \mathbf{e}^3 + \mathbf{v} \cdot \widehat{\text{sk}}$  for some  $\mathbf{e}^3 \in \mathcal{D}_{2\eta}^N$ , where  $(q/p)$  is ignored in the EncMsg execution.
9. Both parties define

$$\tau_{\text{gsw} \rightarrow \text{egsw}} \stackrel{\text{def}}{=} (\mathbf{a}, \mathbf{b}^1 = \mathbf{a} \cdot \text{sk} + \mathbf{e}^1, \mathbf{b}^2 = \mathbf{a} \cdot \widehat{\text{sk}} + \mathbf{e}^2, \mathbf{b}^3 = \mathbf{b}^1 \cdot \widehat{\text{sk}} + \mathbf{e}^3 + \mathbf{v} \cdot \widehat{\text{sk}}).$$

10.  $P_A$  and  $P_B$  execute  $\text{EncMsg}(\widetilde{\mathbf{a}}, M, \text{sk}_A, \text{sk}_B, 0, 0)$  to obtain  $\widetilde{\mathbf{b}} = \widetilde{\mathbf{a}} \cdot \text{sk} + \widetilde{\mathbf{e}} \in \mathcal{R}_q^M$  for some  $\widetilde{\mathbf{e}} \in \mathcal{D}_{2\eta}^M$ .

11. Both parties output  $\text{keypar}' = (\tau_{\text{sk} \rightarrow \widehat{\text{sk}}}, \tau_{\widehat{\text{sk}} \rightarrow \text{sk}}, \tau_{\text{gsw} \rightarrow \text{egsw}}, \widetilde{\mathbf{a}}, \widetilde{\mathbf{b}})$ .

Figure 7: Our two-party key-generation and encryption protocols with active security for RLWE-based SWHE in the  $(\mathcal{F}_{\text{CPC}}, \mathcal{F}_{\text{Rand}})$ -hybrid model (Part 1).

**Protocol  $\Pi_{2\text{PSWHE}}$  (Part 2)**

**Two-party encryption protocol:** A party  $\mathcal{P} \in \{\text{P}_A, \text{P}_B\}$  holds a message  $z = (z_1, \dots, z_t) \in \{0, 1\}^t$ . Both parties perform the following steps to generate the ciphertexts on  $z$ . For each  $i \in [t]$ ,  $\text{P}_A$  and  $\text{P}_B$  do the following:

1.  $\mathcal{P}$  samples a uniform matrix  $\mathbf{R}_i \xleftarrow{\$} \mathcal{R}_2^{N \times M}$ , and calls the Commit command of  $\mathcal{F}_{\text{CPC}}$  to commit  $(z_i, \mathbf{R}_i)$ .
2. Then,  $\mathcal{P}$  computes

$$\llbracket z_i \rrbracket := \text{Flatten} \left( z_i \cdot \mathbf{I}_N + \text{BitDecomp} \left( \begin{bmatrix} \mathbf{R}_i \cdot \tilde{\mathbf{a}} & \mathbf{R}_i \cdot \tilde{\mathbf{b}} \end{bmatrix} \right) \right),$$

and sends  $\llbracket z_i \rrbracket \in \mathcal{R}_2^{N \times N}$  to the other party.

3.  $\text{P}_A$  and  $\text{P}_B$  call the Prove command of  $\mathcal{F}_{\text{CPC}}$  to verify that the following holds:

$$\left\{ (z_i, \mathbf{R}_i) \mid \llbracket z_i \rrbracket = \text{Flatten} \left( z_i \cdot \mathbf{I}_N + \text{BitDecomp} \left( \begin{bmatrix} \mathbf{R}_i \cdot \tilde{\mathbf{a}} & \mathbf{R}_i \cdot \tilde{\mathbf{b}} \end{bmatrix} \right) \right), \right. \\ \left. z_i \in \{0, 1\} \text{ and } \mathbf{R}_i \in \mathcal{R}_2^{N \times M} \right\}.$$

4. Both parties output  $\llbracket z_i \rrbracket$ .

Figure 8: Our two-party key-generation and encryption protocols with active security for RLWE-based SWHE in the  $(\mathcal{F}_{\text{CPC}}, \mathcal{F}_{\text{Rand}})$ -hybrid model (Part 2).

our authenticated BitGC,  $\text{sk}$ ,  $\text{sk}_A$ , and  $\text{sk}_B$  correspond to  $\Delta$ ,  $\Delta_A$ , and  $\Delta_B$ , respectively. This protocol invokes a macro EncMsg shown in Figure 6, realizing the two-party encryption without knowing  $\text{sk}$  for any party. This macro EncMsg is used to generate three ciphertexts  $\tau_{\text{sk} \rightarrow \hat{\text{sk}}}$ ,  $\tau_{\hat{\text{sk}} \rightarrow \text{sk}}$ ,  $\tau_{\text{gsw} \rightarrow \text{egsw}}$  in the key-generation phase, and is also invoked to perform encryption in the encryption phase.

This protocol does not realize the standard key-generation algorithm KeyGen shown in Section 5.1. Instead, it realize the equivalent key-generation algorithm  $(\text{keypar}', \text{sk}) \leftarrow \text{KeyGen}'(\text{params}, \text{sk}_A, \text{sk}_B)$ . Here, KeyGen' is the same as KeyGen, except that it takes  $(\text{sk}_A, \text{sk}_B)$  as input, where  $\text{sk}_A, \text{sk}_B$  are sampled from  $\mathcal{D}_\eta$ . To achieve better efficiency, we adopt the well-known technique [Rot11] of transforming private-key HE to public-key HE in the encryption phase. In particular,  $\text{P}_A$  and  $\text{P}_B$  need to generate  $M \equiv O(n \log^2(q))$  ciphertexts on zero (i.e.,  $(\tilde{\mathbf{a}}, \tilde{\mathbf{b}})$ ) by invoking EncMsg in the key-generation phase, where  $(\tilde{\mathbf{a}}, \tilde{\mathbf{b}})$  serves as the role of public key. In this case, the encryption algorithm is changed as: on input a public key  $(\tilde{\mathbf{a}}, \tilde{\mathbf{b}})$  and a message  $m \in \mathbb{Z}_p$ ,  $\text{Enc}((\tilde{\mathbf{a}}, \tilde{\mathbf{b}}), m)$  samples a uniform matrix  $\mathbf{R} \xleftarrow{\$} \mathcal{R}_2^{N \times M}$ , and then computes a ciphertext  $\llbracket m \rrbracket$  as

$$\text{Flatten} \left( m \cdot \mathbf{I}_N + \text{BitDecomp} \left( \begin{bmatrix} \mathbf{R} \cdot \tilde{\mathbf{a}} & \mathbf{R} \cdot \tilde{\mathbf{b}} \end{bmatrix} \right) \right).$$

Note that the above encryption algorithm can be still applied the main protocol. In addition, if we use an actively secure 2PC to encrypt messages, then we are still able to adopt the original encryption algorithm described in Section 5.1. This is much more expensive, but needs to be run only once. It is straightforward to verify the correctness of protocol  $\Pi_{2\text{PSWHE}}$  (Figures 7 and 8) by working through the whole protocol.

When instantiating  $\mathcal{F}_{\text{CPC}}$ , a prime  $q$  would allow us to use a more efficient protocol. In this case, we need to remove the condition  $p \mid q$  from the rounding lemma (defined in Lemma 2). When revisiting the proof of this lemma (see [BKS19, Appendix B.1]), we find that the condition is not necessary and can be removed. An alternative approach for supporting a prime  $q$  is to use the modulus switching technique [BV11]. Particularly, both parties first execute protocol  $\Pi_{2\text{PSWHE}}$  for a prime  $q$ , and then switch all ciphertexts from a prime  $q$  to a composite number  $q'$ , where  $p \mid q'$  satisfies the condition.

**Analysis of communication and round complexities.** For protocol  $\Pi_{2\text{PSWHE}}$  (Figures 7 and 8), we analyze the communication and rounds in the  $(\mathcal{F}_{\text{CPC}}, \mathcal{F}_{\text{Rand}})$ -hybrid model. We first analyze the macro EncMsg. The communication cost is  $2\ell n \log q$  bits for encrypting  $m \in \mathcal{R}_q^\ell$ , and there are two rounds. In the key-generation phase, macro EncMsg is invoked 6 times, which brings about  $(6N + 2M + 4)n \log q$  bits of communication. Among the 6 times invocations of EncMsg, there are 5 times invocations that can be run in parallel. Thus, we have 4 rounds for two-party key generation. In the encryption phase, we obtain  $tnN^2$  bits of communication for encrypting  $t$  bits of message and only one round.

We only need to compute one level of multiplications (i.e.,  $sk \cdot \hat{sk} \in \mathcal{R}_q$ ) using a 2PC protocol like SPDZ [DPSZ12, DKL<sup>+</sup>13]. Therefore, when instantiating  $\mathcal{F}_{\text{CPC}}$  shown in Section 3.1, the rounds are  $O(1)$ . Together with that  $\mathcal{F}_{\text{Rand}}$  can be instantiated in three rounds, we have constant rounds in total, when considering both key-generation and encryption phases. We can also replace  $\mathcal{F}_{\text{Rand}}$  with the Fiat-Shamir heuristic to further reduce the rounds. As shown in Section 3.1,  $\mathcal{F}_{\text{CPC}}$  can be efficiently realized by a two-party protocol combining VOLE-ZK with SPDZ. The total communication is still small when instantiating  $\mathcal{F}_{\text{CPC}}$ .

**Theorem 2.** *Protocol  $\Pi_{2\text{PSWHE}}$  (Figures 7 and 8) securely computes  $\mathcal{F}_{2\text{PSWHE}}$  in the presence of a static, malicious adversary in the  $(\mathcal{F}_{\text{CPC}}, \mathcal{F}_{\text{Rand}})$ -hybrid model under the RLWE assumption with KDM security.*

The proof of Theorem 2 can be found in Appendix B.2.

## References

- [ADI<sup>+</sup>17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 223–254. Springer, Cham, August 2017.
- [AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in  $\text{NC}^0$ . In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.
- [AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in  $\text{nc}^0$ . *Comput. Complex.*, 17(1):38–69, 2008.
- [AK19] Benny Applebaum and Eliran Kachlon. Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In David Zuckerman, editor, *60th FOCS*, pages 171–179. IEEE Computer Society Press, November 2019.
- [AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Berlin, Heidelberg, May 2014.
- [App12] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 805–816. ACM Press, May 2012.
- [APRR24] Navid Alamati, Guru-Vamsi Policharla, Srinivasan Raghuraman, and Peter Rindal. Improved alternating-moduli PRFs and post-quantum signatures. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 274–308. Springer, Cham, August 2024.

- [ASH<sup>+</sup>20] Jackson Abascal, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. Is the classical GMW paradigm practical? The case of non-interactive actively secure 2PC. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1591–1605. ACM Press, November 2020.
- [Bar96] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th FOCS*, pages 184–193. IEEE Computer Society Press, October 1996.
- [BBC<sup>+</sup>24] Borja Balle, James Bell, Albert Cheu, Adria Gascon, Jonathan Katz, Mariana Raykova, Phillip Schoppmann, and Thomas Steinke. Hash-prune-invert: Improved differentially private heavy-hitter detection in the two-server model. *Cryptology ePrint Archive*, Paper 2024/2024, 2024.
- [BCC<sup>+</sup>24] Dung Bui, Haotian Chu, Geoffroy Couteau, Xiao Wang, Chenkai Weng, Kang Yang, and Yu Yu. An efficient ZK compiler from SIMD circuits to general circuits. *Journal of Cryptology*, 38(10), 2024.
- [BCG<sup>+</sup>19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.
- [BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.
- [BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Berlin, Heidelberg, May 2011.
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.
- [BIP<sup>+</sup>18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 699–729. Springer, Cham, November 2018.
- [BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Cham, May 2019.
- [BLN<sup>+</sup>21] Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High-performance multi-party computation for binary circuits based on oblivious transfer. *Journal of Cryptology*, 34(3):34, July 2021.
- [BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac’n’cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 2021. Springer, Cham.

- [BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Berlin, Heidelberg, April 2012.
- [Bra13] Luís T. A. N. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique - (extended abstract). In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 441–463. Springer, Berlin, Heidelberg, December 2013.
- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.
- [Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
- [CDH<sup>+</sup>23] Sofia Celi, Alex Davidson, Hamed Haddadi, Gonçalo Pestana, and Joe Rowell. DiStefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more. Cryptology ePrint Archive, Report 2023/1063, 2023.
- [CM01] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in  $nc^0$ . In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *MFCS 2001*, volume 2136 of *LNCS*, pages 272–284. Springer, 2001.
- [CWYY23] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 35–67. Springer, Cham, April 2023.
- [DGH<sup>+</sup>21] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 517–547, Virtual Event, August 2021. Springer, Cham.
- [DILO22a] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87. Springer, Cham, August 2022.
- [DILO22b] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Improving line-point zero knowledge: Two multiplications for the price of one. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 829–841. ACM Press, November 2022.
- [DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPICs*, pages 5:1–5:24. Schloss Dagstuhl, July 2021.
- [DKL<sup>+</sup>13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, September 2013.

- [DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.
- [FJN<sup>+</sup>13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 537–556. Springer, Berlin, Heidelberg, May 2013.
- [FJNT15] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. TinyLEGO: An interactive garbling scheme for maliciously secure two-party computation. Cryptology ePrint Archive, Report 2015/309, 2015.
- [FKOS15] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Berlin, Heidelberg, November / December 2015.
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.
- [Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.
- [Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Berlin, Heidelberg, December 2010.
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.
- [HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 18–35. Springer, Berlin, Heidelberg, August 2013.
- [HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, November 2023.
- [HOSS18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT). In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 86–117. Springer, Cham, December 2018.



- [HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, Cham, December 2017.
- [IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Berlin, Heidelberg, August 2008.
- [IZ89] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *30th FOCS*, pages 248–253. IEEE Computer Society Press, October / November 1989.
- [KNR<sup>+</sup>17] Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. DUPLO: Unifying cut-and-choose for garbled circuits. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 3–20. ACM Press, October / November 2017.
- [KOS16] Marcel Keller, Emanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.
- [KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Cham, April / May 2018.
- [KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, Cham, August 2018.
- [KsS12] Benjamin Kreuter, abhi shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In Tadayoshi Kohno, editor, *USENIX Security 2012*, pages 285–300. USENIX Association, August 2012.
- [Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Berlin, Heidelberg, August 2013.
- [LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, Berlin, Heidelberg, May 2007.
- [LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, Berlin, Heidelberg, March 2011.
- [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. Springer, Berlin, Heidelberg, May / June 2010.

- [LWYY25] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: Garbled Circuits with 1 Bit per Gate. In *Advances in Cryptology – EUROCRYPT 2025*. Springer, Heidelberg, Germany, 2025. <https://eprint.iacr.org/2024/1988>.
- [LXY25] Fuchun Lin, Chaoping Xing, and Yizhou Yao. Interactive line-point zero-knowledge with sublinear communication and linear computation. In *Advances in Cryptology – ASIACRYPT 2024*, pages 337–366. Springer Nature Singapore, 2025.
- [LXY25] Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. *Cryptology ePrint Archive*, Paper 2025/169, 2025.
- [MPP20] Andrew Morgan, Rafael Pass, and Antigoni Polychroniadou. Succinct non-interactive secure computation. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 216–245. Springer, Cham, May 2020.
- [NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Berlin, Heidelberg, August 2012.
- [NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, Berlin, Heidelberg, March 2009.
- [NST17] Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *NDSS 2017*. The Internet Society, February / March 2017.
- [QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.
- [Rot11] Ron Rothblum. Homomorphic encryption: From private-key to public-key. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 219–234. Springer, Berlin, Heidelberg, March 2011.
- [RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Cham.
- [sS11] abhi shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, Berlin, Heidelberg, May 2011.
- [sS13] abhi shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 523–534. ACM Press, November 2013.
- [WMK17] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster secure two-party computation in the single-execution setting. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 399–424. Springer, Cham, April / May 2017.

- [WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017.
- [WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56. ACM Press, October / November 2017.
- [WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.
- [WYY<sup>+</sup>22] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. AntMan: Interactive zero-knowledge proofs with sublinear communication. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2901–2914. ACM Press, November 2022.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.
- [YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021.
- [YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1627–1646. ACM Press, November 2020.
- [ZMM<sup>+</sup>20] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1919–1938. ACM Press, November 2020.

## A Other Algorithms for SWHE with Distributed Decryption

The other algorithms of the SWHE scheme with distributed decryption (shown in Section 5.1) are described as follows.

**Extended encryption.** On input two secret keys  $sk, \widehat{sk}$  as well as a message  $m \in \mathbb{Z}_p$ ,  $\widehat{\text{Enc}}(sk, \widehat{sk}, m)$  samples  $a \xleftarrow{\$} \mathcal{R}_q$  and  $e^1, e^2, e^3 \leftarrow \mathcal{D}_\eta$ , and then outputs an eGSW ciphertext

$$\widehat{\tau} = (a, b^1 = a \cdot sk + e^1, b^2 = a \cdot \widehat{sk} + e^2, b^3 = b^1 \cdot \widehat{sk} + e^3 + (q/p) \cdot m \cdot \widehat{sk}).$$

**GSW-to-eGSW conversion**  $[\cdot]_{\text{gsw} \rightarrow \text{egsw}}$ . Given a GSW ciphertext  $\tau = \text{Enc}(sk, m)$  on a message  $m \in \mathbb{Z}_p$  and the GSW-to-eGSW ciphertext  $\tau_{\text{gsw} \rightarrow \text{egsw}} = (a, b^1, b^2, b^3)$ , the GSW-to-eGSW conversion function  $[\tau]_{\text{gsw} \rightarrow \text{egsw}}$  generates an eGSW ciphertext on  $m$  as

$$(a = \langle \mathbf{u}, \mathbf{a} \rangle, b^1 = \langle \mathbf{u}, \mathbf{b}^1 \rangle, b^2 = \langle \mathbf{u}, \mathbf{b}^2 \rangle, b^3 = \langle \mathbf{u}, \mathbf{b}^3 \rangle \approx b^1 \cdot \widehat{sk} + (q/p) \cdot m \cdot \widehat{sk}),$$

where  $\mathbf{u} = \text{BitDecomp}([0 \quad (q/p)]) \cdot \tau$ .

**Key-switching.** Given two key-switching ciphertexts  $\tau_{sk \rightarrow \widehat{sk}}$  and  $\tau_{\widehat{sk} \rightarrow sk}$ , we define two key-switching functions  $[\cdot]_{sk \rightarrow \widehat{sk}}$  and  $[\cdot]_{\widehat{sk} \rightarrow sk}$ . In particular, for any  $y \in \mathcal{R}_p$ , we have

$$[y]_{sk \rightarrow \widehat{sk}} \text{ outputs } [y[1] \cdot b - a \cdot y]_p, \text{ and } [y]_{\widehat{sk} \rightarrow sk} \text{ outputs } [y[1] \cdot \widehat{b} - \widehat{a} \cdot y]_p.$$

Let  $sk_0, \widehat{sk}_0 \in \mathcal{R}_p$  be two uniform elements such that  $\text{LSB}(sk_0) = 0$  and  $\text{LSB}(\widehat{sk}_0) = 0$ . Let  $sk_1 = sk_0 + sk$  and  $\widehat{sk}_1 = \widehat{sk}_0 + \widehat{sk}$  over  $\mathcal{R}_p$ . We have  $[sk_1]_{sk \rightarrow \widehat{sk}} = [sk_0]_{sk \rightarrow \widehat{sk}} + \widehat{sk}$  and  $[\widehat{sk}_1]_{\widehat{sk} \rightarrow sk} = [\widehat{sk}_0]_{\widehat{sk} \rightarrow sk} + sk$  over  $\mathcal{R}_p$ .

**Algorithm  $\widehat{\text{Dec}}$  for correlated-key distributed decryption.** Given a ciphertext  $\tau = \text{Enc}(sk, m)$  as well as two additive secret sharings  $(sk_0, sk_1)$  and  $(sk'_0, sk'_1)$  such that  $\text{LSB}(sk_0) = \text{LSB}(sk'_0) = 0$ ,  $sk_1 = sk_0 + sk$  and  $sk'_1 = sk'_0 + sk$  over  $\mathcal{R}_p$ , for any  $i, j \in \{0, 1\}$ ,  $\widehat{\text{Dec}}(sk_i, sk'_j, \tau)$  performs the following steps:

1. Perform a key-switching operation  $\widehat{sk}_j := [sk'_j]_{sk \rightarrow \widehat{sk}}$ .
2. Convert  $\tau$  into an eGSW ciphertext by computing  $(a, b^1, b^2, b^3) := [\tau]_{\text{gsw} \rightarrow \text{egsw}}$ .
3. Compute  $x := sk_i \cdot \widehat{sk}_j \cdot a - i \cdot \widehat{sk}_j \cdot b^1 - j \cdot sk_i \cdot b^2 + i \cdot j \cdot b^3$  over  $\mathcal{R}_q$ , where  $i = \text{LSB}(sk_i)$  and  $j = \text{LSB}(sk'_j)$ .
4. Set  $y := [x]_p$  and perform another key-switching operation  $z := [y]_{\widehat{sk}_j \rightarrow sk}$ . Output  $z \in \mathcal{R}_p$ .

## B Security Proof

### B.1 Security Proof for the Main Protocol

**Theorem 3** (Theorem 1, restated). *Let  $f$  be a two-party functionality  $\{0, 1\}^{|\mathcal{I}_A|} \times \{0, 1\}^{|\mathcal{I}_B|} \rightarrow \{0, 1\}^{|\mathcal{O}|}$  with  $|\mathcal{I}_A| = |\mathcal{I}_B|$ . Let PRG be a pseudorandom generator, PRF be a pseudorandom function, and  $H$  be a random oracle. Assume that the SWHE scheme is CPA secure under any pair  $(\text{keypar}', \Delta) \leftarrow \text{KeyGen}(\text{params}, \Delta_A, \Delta_B)$ , if at least one party is honest. Then protocol  $\Pi_{2\text{PC}}$  (Figures 4 and 5) securely computes  $f$  with statistical error  $1/p$  in the presence of a static, malicious adversary in the  $(\mathcal{F}_{2\text{PSWHE}}, \mathcal{F}_{\text{Rand}})$ -hybrid model.*

*Proof.* We first consider the case of a malicious  $\mathcal{P}_A$ , and then handle the case of a malicious  $\mathcal{P}_B$ . In each case, we construct a PPT simulator  $\mathcal{S}$  (given access to functionality  $\mathcal{F}_{2PC}$ ), which runs the PPT adversary  $\mathcal{A}$  as a subroutine and emulates functionalities  $\mathcal{F}_{2PSWHE}$  and  $\mathcal{F}_{Rand}$ .

**Malicious  $\mathcal{P}_A$ .**  $\mathcal{S}$  simulates random oracle  $H$  by responding random strings while keeping the consistency of responses. Then,  $\mathcal{S}$  simulates the view of  $\mathcal{A}$  as follows.

PREPROCESSING:  $\mathcal{S}$  simulates the preprocessing phase as follows:

1.  $\mathcal{S}$  emulates the GenKey command of  $\mathcal{F}_{2PSWHE}$  by receiving  $\Delta_A$  from  $\mathcal{A}$ , running  $(keypar', \Delta) \leftarrow \text{KeyGen}'(\text{params}, \Delta_A, \Delta_B)$  with  $\Delta_B \leftarrow \mathcal{D}_\eta$  and sending  $keypar'$  to  $\mathcal{A}$ . Note that  $\mathcal{S}$  never uses  $\Delta$ .
2.  $\mathcal{S}$  emulates  $\mathcal{F}_{Rand}$  by sending a random key  $k \in \{0, 1\}^\lambda$  as well as a uniform element  $U \in \mathcal{R}_p$  with  $\text{LSB}(U) = 0$  to  $\mathcal{A}$ .  $\mathcal{S}$  updates  $\Delta_A$  as  $\Delta_A + U \in \mathcal{R}_p$ .
3.  $\mathcal{S}$  emulates  $\mathcal{F}_{2PSWHE}$  by receiving  $s_A \in \{0, 1\}^\lambda$  from  $\mathcal{A}$  and then sending  $\llbracket s_A \rrbracket$  to  $\mathcal{A}$ . Then,  $\mathcal{S}$  homomorphically computes  $\text{PRG}(\llbracket s_A \rrbracket)$  to obtain  $\{\llbracket \hat{r}_i \rrbracket\}_{i \in \mathcal{I}_A} \cup \{\llbracket r_i \rrbracket\}_{i \in [N]}$ .  $\mathcal{S}$  also computes  $\{\hat{r}_i\}_{i \in \mathcal{I}_A} \cup \{r_i\}_{i \in [N]} := \text{PRG}(s_A)$ .
4.  $\mathcal{S}$  emulates  $\mathcal{F}_{2PSWHE}$  by sending  $\llbracket 0 \rrbracket$  to  $\mathcal{A}$ , and computes  $\{\llbracket \hat{t}_i \rrbracket\}_{i \in \mathcal{I}_B} \cup \{\llbracket t_i \rrbracket\}_{i \in [N]}$  following the protocol specification.

INPUT PROCESSING, CIRCUIT GARBLING AND CIRCUIT EVALUATING:  $\mathcal{S}$  simulates the phases of input processing and circuit garbling as follows:

5. For each  $i \in \mathcal{I}_A$ ,  $\mathcal{S}$  receives  $\sigma_{i,A} \in \{0, 1\}$  from  $\mathcal{A}$ , and computes  $x_i := \hat{r}_i \oplus \sigma_{i,A}$ . Then,  $\mathcal{S}$  sets  $\mathbf{x} := \{x_i\}_{i \in \mathcal{I}_A}$  and sends it to  $\mathcal{F}_{2PC}$ .
6. For each  $i \in \mathcal{I}_B$ ,  $\mathcal{S}$  samples  $\sigma_{i,B} \xleftarrow{\$} \{0, 1\}$  and sends it to  $\mathcal{A}$ .
7. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ ,  $\mathcal{S}$  receives  $d_i \in \{0, 1\}$  from  $\mathcal{A}$ , and computes a correct bit  $d_i^*$  with the  $\mathcal{P}_A$ 's secrets following the protocol specification.
8. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ ,  $\mathcal{S}$  samples  $\tilde{d}_i \xleftarrow{\$} \{0, 1\}$  and sends it to  $\mathcal{A}$ .

OUTPUT PROCESSING AND CONSISTENCY CHECK:  $\mathcal{S}$  simulates this phase as follows:

9. For each  $i \in \mathcal{O}$ ,  $\mathcal{S}$  receives  $\pi_i \in \{0, 1\}$  from  $\mathcal{A}$ , and also computes a correct bit  $\pi_i^*$  with the  $\mathcal{P}_A$ 's secrets following the protocol specification.
10.  $\mathcal{S}$  emulates  $\mathcal{F}_{Rand}$  by sending random challenges  $\{\chi_i\}_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}}$  and  $\{\chi'_i\}_{i \in \mathcal{O}}$  from  $\mathbb{Z}_p$  to  $\mathcal{A}$ . Then,  $\mathcal{S}$  computes the ciphertext  $\tau$  following the protocol specification.
11.  $\mathcal{S}$  receives  $V_A \in \{0, 1\}^\lambda$  from  $\mathcal{A}$ . If there exists some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  such that  $d_i \neq d_i^*$ , or some  $i \in \mathcal{O}$  satisfying  $\pi_i \neq \pi_i^*$ , or  $V_A \neq H(\text{Dec}(\Delta_A, \tau), \tau)$ ,  $\mathcal{S}$  sends abort to  $\mathcal{F}_{2PC}$  and aborts. Otherwise,  $\mathcal{S}$  accepts the execution.

We use a sequence of hybrids to show that the real-world execution and ideal-world execution are computationally indistinguishable.

**Hybrid 0.** This hybrid (denoted by  $\mathcal{G}_0$ ) is the real-world execution.

**Hybrid 1.** This hybrid (denoted by  $\mathcal{G}_1$ ) is the same as  $\mathcal{G}_0$ , except that emulating  $\mathcal{F}_{2PSWHE}$  and  $\mathcal{F}_{Rand}$ , and simulating random oracle  $H$  honestly.

It is obvious that  $\mathcal{G}_1$  is perfectly indistinguishable from  $\mathcal{G}_0$ . Hybrid  $\mathcal{G}_1$  can obtain all secrets of  $\mathcal{P}_A$  by emulating  $\mathcal{F}_{2\text{PSWHE}}$ .

**Hybrid 2.** This hybrid (denoted by  $\mathcal{G}_2$ ) is the same as  $\mathcal{G}_1$ , except for the following differences:

1. Computing correct bits  $d_i^*$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  and  $\pi_i^*$  for all  $i \in \mathcal{O}$  following the simulation of  $\mathcal{S}$ .
2. Performing the final check by verifying whether  $d_i = d_i^*$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ ,  $\pi_i = \pi_i^*$  for all  $i \in \mathcal{O}$  and  $V_A = H(\text{Dec}(\Delta_A, \tau), \tau)$ .

The only difference between  $\mathcal{G}_1$  and  $\mathcal{G}_2$  is the manner of consistency check. In  $\mathcal{G}_1$ , the check is done by verifying if  $V_A = H(\text{Dec}(\Delta_B, \tau), \tau)$ . In  $\mathcal{G}_2$ , the check in step 2 is used. Below, we prove that the difference is bounded by  $1/p + \text{negl}(\lambda)$ .

In  $\mathcal{G}_1$ , if  $V_A = H(\text{Dec}(\Delta_B, \tau), \tau)$ , we prove that  $\tau$  encrypts zero under the assumption that the SWHE scheme is CPA secure. Suppose that  $\tau$  encrypts a non-zero element  $E$  but the check still passes in  $\mathcal{G}_1$ . In this case,  $\mathcal{A}$  must make a query  $(\text{Dec}(\Delta_B, \tau), \tau)$  to random oracle  $H$ . We construct a PPT algorithm  $\mathcal{B}$  to break the CPA security of SWHE. In particular,  $\mathcal{B}$  behaves just like in  $\mathcal{G}_1$  and simulates random oracle  $H$ . Then  $\mathcal{B}$  retrieves  $\text{Dec}(\Delta_B, \tau)$  from the queries of  $H$  with non-negligible probability. Finally,  $\mathcal{B}$  computes  $\text{Dec}(\Delta_B, \tau) - \text{Dec}(\Delta_A, \tau) = \text{Dec}(\Delta, \tau) = E \cdot \Delta$ , and then recovers  $\Delta$  where  $\mathcal{B}$  knows  $E$ . In conclusion, we have that  $\tau$  encrypts zero if  $V_A = H(\text{Dec}(\Delta_B, \tau), \tau)$ , except with probability  $\text{negl}(\lambda)$ . Let  $e_i \stackrel{\text{def}}{=} v_i^* - v_i$  for  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  and  $e'_i \stackrel{\text{def}}{=} \hat{\pi}_i - \pi_i$  for  $i \in \mathcal{O}$ , where  $\hat{\pi}_i \stackrel{\text{def}}{=} r_i \oplus d_i$ . If  $V_A = H(\text{Dec}(\Delta_B, \tau), \tau)$ , we have

$$\sum_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}} \chi_i \cdot e_i + \sum_{i \in \mathcal{O}} \chi'_i \cdot e'_i = 0.$$

Note that  $e_i, e'_i \in \{-1, 0, 1\}$ . Since the challenges  $\chi_i, \chi'_i \in \mathbb{Z}_p$  are sampled uniformly after  $e_i, e'_i$  have been defined, we obtain  $e_i = 0$  and  $e'_i = 0$ , except with probability  $1/p$ . This means that  $v_i = v_i^*$  for  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  and  $\pi_i = \hat{\pi}_i$  for  $i \in \mathcal{O}$ . Since  $v_i = r_i \oplus d_i \oplus t_i \oplus \tilde{d}_i$ , we obtain  $v_i = v_i^*$  if and only if  $d_i = d_i^*$ , where both  $v_i^*$  and  $d_i^*$  are correct. Thus, we have  $\pi_i^* = \hat{\pi}_i = \pi_i$  for each  $i \in \mathcal{O}$ . According to the correctness analysis shown in Section 4.2, we know if  $d_i = d_i^*$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  and  $\pi_i = \pi_i^*$  for each  $i \in \mathcal{O}$ , then  $V_A = H(\text{Dec}(\Delta_B, \tau), \tau)$ . In other words, except with probability  $1/p + \text{negl}(\lambda)$ , the check of  $V_A = H(\text{Dec}(\Delta_B, \tau), \tau)$  is equivalent to checking  $d_i = d_i^*$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$ ,  $\pi_i = \pi_i^*$  for all  $i \in \mathcal{O}$  and  $V_A = H(\text{Dec}(\Delta_A, \tau), \tau)$ .

**Hybrid 3.** This hybrid (denoted by  $\mathcal{G}_3$ ) is the same as  $\mathcal{G}_2$ , except that replacing  $\llbracket s_B \rrbracket$  with  $\llbracket 0 \rrbracket$ .

We prove that  $\mathcal{G}_3$  is indistinguishable from  $\mathcal{G}_2$  using a reduction to the CPA security of the SWHE scheme. Specifically, if the difference between  $\mathcal{G}_2$  and  $\mathcal{G}_3$  is noticeable, we construct a PPT algorithm  $\mathcal{B}$  to break the CPA security of SWHE.  $\mathcal{B}$  chooses  $s_B$  and 0 as two messages, and obtains the challenge ciphertext  $\tau^*$ . Then  $\mathcal{B}$  behaves just like in  $\mathcal{G}_2$ , except that using  $\tau^*$  as  $\llbracket s_B \rrbracket$ . If  $\tau^*$  encrypts  $s_B$ ,  $\mathcal{B}$  behaves exactly in  $\mathcal{G}_2$ . If  $\tau^*$  is the encryption of 0,  $\mathcal{B}$  behaves exactly in  $\mathcal{G}_3$ .

**Hybrid 4.** This hybrid (denoted by  $\mathcal{G}_4$ ) is the same as  $\mathcal{G}_3$ , except for sampling  $\sigma_{i,B} \in \{0, 1\}$  for each  $i \in \mathcal{I}_B$  and  $\tilde{d}_i \in \{0, 1\}$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  uniformly at random.

We use the pseudorandomness of PRG outputs to bound the difference between  $\mathcal{G}_3$  and  $\mathcal{G}_4$ . Based on that PRG is a pseudorandom generator,  $\hat{t}_i$  for all  $i \in \mathcal{I}_B$  and  $t_i$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  are computationally indistinguishable from uniform bits. Therefore, in  $\mathcal{G}_3$ , both  $\sigma_{i,B}$  for each  $i \in \mathcal{I}_B$  and  $\tilde{d}_i \in \{0, 1\}$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{G}$  are computationally indistinguishable from random bits. Thus,  $\mathcal{G}_4$  is computationally indistinguishable from  $\mathcal{G}_3$ .

**Hybrid 5.** This hybrid (denoted by  $\mathcal{G}_5$ ) is the same as  $\mathcal{G}_4$ , except that extracting  $x_i := \hat{r}_i \oplus \sigma_{i,A}$  for each  $i \in \mathcal{I}_A$ , and sending  $\mathbf{x} := \{x_i\}_{i \in \mathcal{I}_A}$  to  $\mathcal{F}_{2\text{PC}}$ . This is the ideal-world execution.

It is straightforward to see that the extraction of  $P_A$ 's input is perfect. In  $\mathcal{G}_5$ ,  $P_B$  receives the output  $f(\mathbf{x}, \mathbf{y})$ , if the execution does not abort. In hybrid  $\mathcal{G}_4$ , we have that  $d_i = d_i^*$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$  and  $\pi_i = \pi_i^*$  for all  $i \in O$ , if the protocol execution does not abort. Thus, honest party  $P_B$  would obtain the output  $f(\mathbf{x}, \mathbf{y})$  in  $\mathcal{G}_4$ , according to the correctness analysis shown in Section 4.2, under the assumption that PRF is a pseudorandom function. Overall,  $\mathcal{G}_5$  and  $\mathcal{G}_4$  are computationally indistinguishable.

In conclusion, the joint distribution of  $\mathcal{S}$  and honest  $P_B$ 's outputs in the ideal-world execution is computationally indistinguishable from that of the outputs of  $\mathcal{A}$  and honest  $P_B$  in the real-world execution.

**Malicious  $P_B$ .**  $\mathcal{S}$  simulates random oracle  $H$  by responding random strings while keeping the consistency of responses. Then,  $\mathcal{S}$  simulates the view of  $\mathcal{A}$  as follows.

PREPROCESSING:  $\mathcal{S}$  simulates the preprocessing phase as follows:

1.  $\mathcal{S}$  emulates the GenKey command of  $\mathcal{F}_{2\text{PSWHE}}$  by receiving  $\Delta_B$  from  $\mathcal{A}$ , running  $(\text{keypar}', \Delta) \leftarrow \text{KeyGen}'(\text{params}, \Delta_A, \Delta_B)$  with  $\Delta_A \leftarrow \mathcal{D}_\eta$  and sending  $\text{keypar}'$  to  $\mathcal{A}$ . As such,  $\mathcal{S}$  never uses  $\Delta$ .
2.  $\mathcal{S}$  emulates  $\mathcal{F}_{\text{Rand}}$  by sending a random key  $\mathbf{k} \in \{0, 1\}^\lambda$  as well as a uniform element  $U \in \mathcal{R}_p$  with  $\text{LSB}(U) = 0$  to  $\mathcal{A}$ .  $\mathcal{S}$  updates  $\Delta_B$  as  $\Delta_B + U \in \mathcal{R}_p$ .
3.  $\mathcal{S}$  emulates  $\mathcal{F}_{2\text{PSWHE}}$  by sending  $\llbracket 0 \rrbracket$  to  $\mathcal{A}$ . Then,  $\mathcal{S}$  computes the ciphertexts  $\{\llbracket \hat{r}_i \rrbracket\}_{i \in \mathcal{I}_A} \cup \{\llbracket r_i \rrbracket\}_{i \in [N]}$  following the protocol specification.
4.  $\mathcal{S}$  emulates  $\mathcal{F}_{2\text{PSWHE}}$  by receiving  $\mathbf{s}_B \in \{0, 1\}^\lambda$  from  $\mathcal{A}$  and then sending  $\llbracket \mathbf{s}_B \rrbracket$  to  $\mathcal{A}$ . Then,  $\mathcal{S}$  homomorphically computes  $\widetilde{\text{PRG}}(\llbracket \mathbf{s}_B \rrbracket)$  to obtain  $\{\llbracket \hat{t}_i \rrbracket\}_{i \in \mathcal{I}_B} \cup \{\llbracket t_i \rrbracket\}_{i \in [N]}$ .  $\mathcal{S}$  also computes  $\{\hat{t}_i\}_{i \in \mathcal{I}_B} \cup \{t_i\}_{i \in [N]} := \text{PRG}(\mathbf{s}_B)$ .

INPUT PROCESSING, CIRCUIT GARBLING AND CIRCUIT EVALUATING:  $\mathcal{S}$  simulates the phases of input processing and circuit garbling as follows:

5. For each  $i \in \mathcal{I}_A$ ,  $\mathcal{S}$  samples  $\sigma_{i,A} \xleftarrow{\$} \{0, 1\}$  and sends it to  $\mathcal{A}$ .
6. For each  $i \in \mathcal{I}_B$ ,  $\mathcal{S}$  receives  $\sigma_{i,B} \in \{0, 1\}$  from  $\mathcal{A}$ , and computes  $y_i := \hat{t}_i \oplus \sigma_{i,B}$ . Then,  $\mathcal{S}$  sets  $\mathbf{y} := \{y_i\}_{i \in \mathcal{I}_B}$ , sends it to  $\mathcal{F}_{2\text{PC}}$  and obtains  $\mathbf{z}^* = f(\mathbf{x}, \mathbf{y})$ , i.e.,  $z_i^* \in \{0, 1\}$  for all  $i \in O$ .
7. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ ,  $\mathcal{S}$  samples  $d_i \xleftarrow{\$} \{0, 1\}$ , and sends it to  $\mathcal{A}$ .
8. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ ,  $\mathcal{S}$  receives  $\tilde{d}_i \in \{0, 1\}$  from  $\mathcal{A}$ , and also computes a correct bit  $\tilde{d}_i^*$  with the  $P_B$ 's secrets following the protocol specification.

OUTPUT PROCESSING AND CONSISTENCY CHECK:  $\mathcal{S}$  simulates this phase as follows:

9. For each  $i \in O$ ,  $\mathcal{S}$  computes  $\pi_i := z_i^* \oplus \text{LSB}(W_{i,B})$  and sends it to  $\mathcal{A}$ , where  $W_{i,B}$  is computed following the protocol specification.
10.  $\mathcal{S}$  emulates  $\mathcal{F}_{\text{Rand}}$  by sending random challenges  $\{\chi_i\}_{i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G}$  and  $\{\chi'_i\}_{i \in O}$  from  $\mathbb{Z}_p$  to  $\mathcal{A}$ . Then,  $\mathcal{S}$  computes  $\tau$  following the protocol specification.
11. If  $\tilde{d}_i \neq \tilde{d}_i^*$  for some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ ,  $\mathcal{S}$  samples  $V_A \xleftarrow{\$} \{0, 1\}^\lambda$  and sends it to  $\mathcal{A}$ . Otherwise,  $\mathcal{S}$  sends  $V_A := H(\text{Dec}(\Delta_B, \tau), \tau)$  to  $\mathcal{A}$ .

We use a sequence of hybrids to show that the real-world execution and ideal-world execution are computationally indistinguishable.

**Hybrid 0.** This hybrid (denoted by  $\mathcal{G}_0$ ) is the real-world execution.

**Hybrid 1.** This hybrid (denoted by  $\mathcal{G}_1$ ) is the same as  $\mathcal{G}_0$ , except that emulating  $\mathcal{F}_{2\text{PSWHE}}$  and  $\mathcal{F}_{\text{Rand}}$ , and simulating random oracle  $H$  honestly.

It is easy to see that  $\mathcal{G}_1$  is perfectly indistinguishable from  $\mathcal{G}_0$ . Hybrid  $\mathcal{G}_1$  can obtain all  $P_B$ 's secrets by emulating  $\mathcal{F}_{2\text{PSWHE}}$ .

**Hybrid 2.** This hybrid (denoted by  $\mathcal{G}_2$ ) is the same as  $\mathcal{G}_1$ , except for the following differences:

1. For each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ , computing a correct bit  $\tilde{d}_i^*$  following the simulation by  $\mathcal{S}$ .
2. If  $\tilde{d}_i \neq \tilde{d}_i^*$  for some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ ,  $\mathcal{S}$  samples  $V_A \xleftarrow{\$} \{0, 1\}^\lambda$  and sends it to  $\mathcal{A}$ . Otherwise,  $\mathcal{S}$  sends  $V_A := H(\text{Dec}(\Delta_B, \tau), \tau)$  to  $\mathcal{A}$ .

The only difference between  $\mathcal{G}_1$  and  $\mathcal{G}_2$  is how to generate  $V_A$ . In  $\mathcal{G}_1$ ,  $V_A$  is always computed as  $H(\text{Dec}(\Delta_B, \tau), \tau)$ . In  $\mathcal{G}_2$ , either  $V_A$  is sampled at random if  $\tilde{d}_i \neq \tilde{d}_i^*$  for some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ , or  $V_A$  is computed with  $\Delta_B$  otherwise. Following a similar analysis used in the case of malicious  $P_A$ , we can prove if  $\tilde{d}_i \neq \tilde{d}_i^*$  for some  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ , then  $\tau$  encrypts a non-zero element  $E$ , except with probability  $1/p$ . In this case, we obtain that  $V_A = H(\text{Dec}(\Delta_B, \tau), \tau)$  is computationally indistinguishable from a uniform string under the assumption that the SWHE scheme is CPA secure, following the same analysis used in the case of malicious  $P_A$ . If  $\tilde{d}_i = \tilde{d}_i^*$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$ , we have  $V_A = V_B = H(\text{Dec}(\Delta_B, \tau), \tau)$  according to the correctness analysis shown in Section 4.2 under the assumption that PRF is a pseudorandom function.

**Hybrid 3.** This hybrid (denoted by  $\mathcal{G}_3$ ) is the same as  $\mathcal{G}_2$ , except that replacing  $\llbracket s_A \rrbracket$  with  $\llbracket 0 \rrbracket$ .

We bound the difference between  $\mathcal{G}_2$  and  $\mathcal{G}_3$  using a reduction to the CPA security of the SWHE scheme. Specifically, if the difference between  $\mathcal{G}_2$  and  $\mathcal{G}_3$  is non-negligible, we construct a PPT algorithm  $\mathcal{B}$  to break the CPA security of SWHE.  $\mathcal{B}$  chooses  $s_A$  and 0 as two messages, and obtains the challenge ciphertext  $\tau^*$ . Then  $\mathcal{B}$  behaves just like in  $\mathcal{G}_2$ , except that using  $\tau^*$  as  $\llbracket s_A \rrbracket$ . If  $\tau^*$  encrypts  $s_A$ ,  $\mathcal{B}$  behaves exactly in  $\mathcal{G}_2$ . If  $\tau^*$  is the encryption of 0,  $\mathcal{B}$  behaves exactly in  $\mathcal{G}_3$ .

**Hybrid 4.** This hybrid (denoted by  $\mathcal{G}_4$ ) is the same as  $\mathcal{G}_3$ , except for sampling  $\sigma_{i,A} \in \{0, 1\}$  for each  $i \in \mathcal{I}_A$  and  $d_i \in \{0, 1\}$  for each  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$  uniformly at random.

We use the pseudorandomness of PRG outputs to bound the difference between  $\mathcal{G}_3$  and  $\mathcal{G}_4$ . Based on that PRG is a pseudorandom generator,  $\hat{r}_i$  for all  $i \in \mathcal{I}_A$  and  $r_i$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$  are computationally indistinguishable from uniform bits. Therefore, in  $\mathcal{G}_2$ ,  $\sigma_{i,A}$  for all  $i \in \mathcal{I}_A$  and  $d_i \in \{0, 1\}$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$  are computationally indistinguishable from uniformly random bits. Thus,  $\mathcal{G}_4$  is computationally indistinguishable from  $\mathcal{G}_3$ .

**Hybrid 5.** This hybrid (denoted by  $\mathcal{G}_5$ ) is the same as  $\mathcal{G}_4$ , except for the following differences:

1. Computing  $y_i := \hat{t}_i \oplus \sigma_{i,B}$  for each  $i \in \mathcal{I}_B$ , sending  $\mathbf{y} = \{y_i\}_{i \in \mathcal{I}_B}$  to  $\mathcal{F}_{2\text{PC}}$ , and receiving  $\{z_i^*\}_{i \in O} = f(\mathbf{x}, \mathbf{y})$ .
2. For each  $i \in O$ , computing  $\pi_i := z_i^* \oplus \text{LSB}(W_{i,B})$  and sending it to  $\mathcal{A}$ .

This is the ideal-world execution.

It is easy to see that the extraction of  $P_B$ 's input is perfect. If the protocol execution does not abort,  $\mathcal{A}$  would obtain the circuit output  $\{z_i^*\}_{i \in O}$  in  $\mathcal{G}_5$ . In hybrid  $\mathcal{G}_4$ , if the protocol execution does not abort, we have that  $\tilde{d}_i$  for all  $i \in \mathcal{I}_A \cup \mathcal{I}_B \cup G$  are correct. Thus,  $\mathcal{A}$  would obtain the output  $f(\mathbf{x}, \mathbf{y})$ , according to the correctness analysis shown in Section 4.2, under the assumption that PRF is a pseudorandom function. Therefore,  $\mathcal{G}_5$  is computationally indistinguishable from  $\mathcal{G}_4$ .

In conclusion, the joint distribution of the outputs of  $\mathcal{S}$  and honest party  $P_A$  in the ideal-world execution is computationally indistinguishable from that of the outputs of  $\mathcal{A}$  and honest party  $P_A$  in the real-world execution, which completes the proof.  $\square$



## B.2 Security Proof of Protocol $\Pi_{2\text{PSWHE}}$

**Theorem 4** (Theorem 2, restated). *Protocol  $\Pi_{2\text{PSWHE}}$  (Figures 7 and 8) securely computes  $\mathcal{F}_{2\text{PSWHE}}$  in the presence of a static, malicious adversary in the  $(\mathcal{F}_{\text{CPC}}, \mathcal{F}_{\text{Rand}})$ -hybrid model under the RLWE assumption with KDM security.*

*Proof.* The roles of  $P_A$  and  $P_B$  are symmetric in the protocol  $\Pi_{2\text{PSWHE}}$ . Thus, we do not distinguish the case of malicious  $P_A$  from that of malicious  $P_B$ . Instead, we prove both cases in a unified way. Let  $P_C$  denote the malicious party controlled by the adversary and  $P_H$  be the honest party, where  $C, H \in \{A, B\}$ . For any PPT adversary  $\mathcal{A}$ , we construct a PPT simulator  $\mathcal{S}$  with access to functionality  $\mathcal{F}_{2\text{PSWHE}}$ .  $\mathcal{S}$  runs  $\mathcal{A}$  as a subroutine, emulates  $\mathcal{F}_{\text{CPC}}$  and  $\mathcal{F}_{\text{Rand}}$ , and simulates the view of  $\mathcal{A}$  as follows. We first show how to simulate the macro  $\text{EncMsg}$ .

MACRO  $\text{EncMsg}(\mathbf{a}, \ell, \text{sk}_A, \text{sk}_B, \mathbf{m}_A, \mathbf{m}_B)$ . The inputs  $\mathbf{a} \in \mathcal{R}_q^\ell$  and  $\ell \in \mathbb{N}$  are public.  $\mathcal{S}$  simulates Macro  $\text{EncMsg}$  as follows:

1. For a malicious party  $P_C$ ,  $\mathcal{S}$  receives  $\mathbf{b}_C \in \mathcal{R}_q^\ell$  from  $\mathcal{A}$ .  $\mathcal{S}$  then emulates the Prove command of  $\mathcal{F}_{\text{CPC}}$  to perform the check following the protocol specification.
2. For an honest party  $P_H$ ,  $\mathcal{S}$  samples  $\mathbf{b}_H \xleftarrow{\$} \mathcal{R}_q^\ell$  and then sends it to  $\mathcal{A}$ .  $\mathcal{S}$  then emulates the Prove command of  $\mathcal{F}_{\text{CPC}}$  to convince  $\mathcal{A}$  that the relationship specified in the protocol description holds.
3.  $\mathcal{S}$  locally computes  $\mathbf{b} := \mathbf{b}_B - \mathbf{b}_A \in \mathcal{R}_q^\ell$ .

TWO-PARTY KEY-GENERATION PROTOCOL.  $\mathcal{S}$  simulates the two-party key generation protocol as follows:

1.  $\mathcal{S}$  emulates the Commit command of  $\mathcal{F}_{\text{CPC}}$  by receiving  $\text{sk}_C \in \mathcal{R}_p$  and  $\widehat{\text{sk}}_C \in \mathcal{R}_p$  from  $\mathcal{A}$ .  $\mathcal{S}$  emulates the Prove command of  $\mathcal{F}_{\text{CPC}}$  to perform the check following the protocol specification. Then,  $\mathcal{S}$  sends  $(\text{sk}_C, \widehat{\text{sk}}_C)$  to  $\mathcal{F}_{2\text{PSWHE}}$ .
2. For the secret keys held by honest party  $P_H$ ,  $\mathcal{S}$  emulates functionality  $\mathcal{F}_{\text{CPC}}$  in a trivial way.
3.  $\mathcal{S}$  emulates  $\mathcal{F}_{\text{Rand}}$  by sending random elements  $a, \widehat{a} \in \mathcal{R}_q$  and uniform vectors  $\mathbf{a} \in \mathcal{R}_q^N$  and  $\widetilde{\mathbf{a}} \in \mathcal{R}_q^M$  to  $\mathcal{A}$ .
4.  $\mathcal{S}$  simulates  $\text{EncMsg}(a, 1, \text{sk}_A, \text{sk}_B, \widehat{\text{sk}}_A, \widehat{\text{sk}}_B)$  to obtain  $\tau_{\text{sk} \rightarrow \widehat{\text{sk}}}$ .
5.  $\mathcal{S}$  simulates  $\text{EncMsg}(\widehat{a}, 1, \widehat{\text{sk}}_A, \widehat{\text{sk}}_B, \text{sk}_A, \text{sk}_B)$  to obtain  $\tau_{\widehat{\text{sk}} \rightarrow \text{sk}}$ .
6.  $\mathcal{S}$  simulates  $\text{EncMsg}(\mathbf{a}, N, \text{sk}_A, \text{sk}_B, 0, 0)$  and  $\text{EncMsg}(\mathbf{a}, N, \widehat{\text{sk}}_A, \widehat{\text{sk}}_B, 0, 0)$  to generate  $\mathbf{b}^1, \mathbf{b}^2 \in \mathcal{R}_q^N$ .
7.  $\mathcal{S}$  emulates the Compute command of  $\mathcal{F}_{\text{CPC}}$  by receiving  $x_C \in \mathcal{R}_q$  from  $\mathcal{A}$ , and locally computes  $\mathbf{m}_C \in \mathcal{R}_q^N$  following the protocol specification.
8.  $\mathcal{S}$  simulates  $\text{EncMsg}(\mathbf{b}^1, N, \text{sk}_A, \text{sk}_B, \mathbf{m}_A, \mathbf{m}_B)$  to obtain  $\mathbf{b}^3$ .
9.  $\mathcal{S}$  defines  $\tau_{\text{gsw} \rightarrow \text{egsw}} \stackrel{\text{def}}{=} (\mathbf{a}, \mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3)$ .
10.  $\mathcal{S}$  simulates  $\text{EncMsg}(\widetilde{\mathbf{a}}, M, \text{sk}_A, \text{sk}_B, 0, 0)$  to obtain  $\widetilde{\mathbf{b}} \in \mathcal{R}_q^M$ .

TWO-PARTY ENCRYPTION PROTOCOL.  $\mathcal{S}$  simulates the two-party encryption protocol in two cases:

Case 1: If honest party  $P_H$  encrypts  $z = (z_1, \dots, z_t) \in \{0, 1\}^t$ , then  $\mathcal{S}$  simulates as follows:

1.  $\mathcal{S}$  emulates the Commit command of  $\mathcal{F}_{\text{CPC}}$  in a trivial way.
2. For each  $i \in [t]$ ,  $\mathcal{S}$  samples  $\mathbf{a}_i, \mathbf{b}_i \xleftarrow{\$} \mathcal{R}_q^N$  and computes

$$\llbracket 0 \rrbracket := \text{Flatten} (0 \cdot \mathbf{I}_N + \text{BitDecomp} ([\mathbf{a}_i \ \mathbf{b}_i])) .$$

Then,  $\mathcal{S}$  sends these ciphertexts on zero to  $\mathcal{A}$ .

3.  $\mathcal{S}$  emulates the Prove command of  $\mathcal{F}_{\text{CPC}}$  to convince  $\mathcal{A}$  that the relationship specified in the protocol description holds.

Case 2: If malicious party  $\mathcal{P}_C$  encrypts  $\mathbf{z} = (z_1, \dots, z_t) \in \{0, 1\}^t$ , then  $\mathcal{S}$  proceeds as follows:

1.  $\mathcal{S}$  emulates the Commit command of  $\mathcal{F}_{\text{CPC}}$  by receiving  $(z_i, \mathbf{R}_i) \in \{0, 1\} \times \mathcal{R}_2^{N \times M}$  from  $\mathcal{A}$ . Then,  $\mathcal{S}$  sends  $\{(z_i, \mathbf{R}_i)\}_{i \in [t]}$  to  $\mathcal{F}_{2\text{PSWHE}}$ .
2. For each  $i \in [t]$ ,  $\mathcal{S}$  receives  $\llbracket z_i \rrbracket \in \mathcal{R}_2^{N \times N}$  from  $\mathcal{A}$ .
3.  $\mathcal{S}$  emulates the Prove command of  $\mathcal{F}_{\text{CPC}}$  to perform the check following the protocol specification.

We use a sequence of hybrids to prove the computational indistinguishability between the real-world execution and ideal-world execution.

**Hybrid 0.** This hybrid (denoted by  $\mathcal{G}_0$ ) is the real-world execution.

**Hybrid 1.** This hybrid (denoted by  $\mathcal{G}_1$ ) is the same as  $\mathcal{G}_0$ , except that emulating  $\mathcal{F}_{\text{CPC}}$  and  $\mathcal{F}_{\text{Rand}}$ .

It is obvious that  $\mathcal{G}_0$  and  $\mathcal{G}_1$  have the identical distribution. In this hybrid, all secrets held by malicious party  $\mathcal{P}_C$  are extracted by  $\mathcal{S}$ .

**Hybrid 2.** This hybrid (denoted by  $\mathcal{G}_2$ ) is the same as  $\mathcal{G}_1$ , except that replacing  $\tau_{\text{sk} \rightarrow \widehat{\text{sk}}}, \tau_{\widehat{\text{sk}} \rightarrow \text{sk}}, \mathbf{b}^1, \mathbf{b}^2$ , and  $\widetilde{\mathbf{b}}$  with uniform vectors over  $\mathcal{R}_q$  in the key-generation protocol.

It is straightforward to bound the difference between  $\mathcal{G}_1$  and  $\mathcal{G}_2$  using a reduction to the RLWE assumption with KDM security. For macro EncMsg, even if  $\mathcal{A}$  chooses  $\mathbf{b}_C$  with  $C \in \{A, B\}$ , the resulting vector  $\mathbf{b} = \mathbf{b}_B - \mathbf{b}_A$  is still computationally indistinguishable from a uniform vector, as  $\mathbf{b}_H$  with  $H \in \{A, B\}$  is computationally indistinguishable from a uniform vector under the RLWE assumption with KDM security. Thus,  $\mathcal{G}_2$  is computationally indistinguishable from  $\mathcal{G}_1$ .

**Hybrid 3.** This hybrid (denoted by  $\mathcal{G}_3$ ) is the same as  $\mathcal{G}_2$ , except that replacing  $\mathbf{b}^3$  with a uniform vector in  $\mathcal{R}_q^N$  in the key-generation protocol.

It is easy to bound the difference between  $\mathcal{G}_2$  and  $\mathcal{G}_3$  using a reduction to the RLWE assumption with KDM security. Thus,  $\mathcal{G}_3$  is computationally indistinguishable from  $\mathcal{G}_2$ .

**Hybrid 4.** This hybrid (denoted by  $\mathcal{G}_4$ ) is the same as  $\mathcal{G}_3$ , except that if an honest party  $\mathcal{P}_H$  encrypts a message  $\mathbf{z} = (z_1, \dots, z_t) \in \{0, 1\}^t$ , then for each  $i \in [t]$ , replacing  $(\mathbf{R}_i \cdot \widetilde{\mathbf{a}}, \mathbf{R}_i \cdot \widetilde{\mathbf{b}})$  with uniform vectors  $(\mathbf{a}_i, \mathbf{b}_i)$  from  $(\mathcal{R}_q^N)^2$  and computing  $\llbracket z_i \rrbracket$  as a ciphertext on zero

$$\llbracket 0 \rrbracket = \text{Flatten} (0 \cdot \mathbf{I}_N + \text{BitDecomp} ([\mathbf{a}_i \ \mathbf{b}_i])) .$$

By applying the leftover hash lemma [IZ89], for each  $i \in [t]$ ,  $(\mathbf{R}_i \cdot \tilde{\mathbf{a}}, \mathbf{R}_i \cdot \tilde{\mathbf{b}})$  is indistinguishable from a uniform vector  $(\mathbf{a}_i, \mathbf{b}_i)$  in  $(\mathcal{R}_q^N)^2$ . Using the properties of Flatten and BitDecomp, we have:

$$\begin{aligned}
\llbracket z_i \rrbracket &= \text{Flatten} \left( z_i \cdot \mathbf{I}_N + \text{BitDecomp} \left( \begin{bmatrix} \mathbf{R}_i \cdot \tilde{\mathbf{a}} & \mathbf{R}_i \cdot \tilde{\mathbf{b}} \end{bmatrix} \right) \right) \\
&= \text{BitDecomp} \left( \text{BitDecomp}^{-1} \left( z_i \cdot \mathbf{I}_N + \text{BitDecomp} \left( \begin{bmatrix} \mathbf{R}_i \cdot \tilde{\mathbf{a}} & \mathbf{R}_i \cdot \tilde{\mathbf{b}} \end{bmatrix} \right) \right) \right) \\
&= \text{BitDecomp} \left( \text{BitDecomp}^{-1} (z_i \cdot \mathbf{I}_N) + \begin{bmatrix} \mathbf{R}_i \cdot \tilde{\mathbf{a}} & \mathbf{R}_i \cdot \tilde{\mathbf{b}} \end{bmatrix} \right) \\
&\approx \text{BitDecomp} \left( \text{BitDecomp}^{-1} (z_i \cdot \mathbf{I}_N) + \begin{bmatrix} \mathbf{a}_i & \mathbf{b}_i \end{bmatrix} \right) .
\end{aligned}$$

Therefore,  $\llbracket z_i \rrbracket$  is indistinguishable from  $\text{BitDecomp} \left( \begin{bmatrix} \mathbf{a}_i & \mathbf{b}_i \end{bmatrix} \right) = \llbracket 0 \rrbracket$ , as  $(\mathbf{a}_i, \mathbf{b}_i)$  is uniform.

**Hybrid 5.** This hybrid (denoted by  $\mathcal{G}_5$ ) is the same as  $\mathcal{G}_4$ , except that sending  $(\text{sk}_C, \widehat{\text{sk}}_C)$  along with  $\{(z_i, \mathbf{R}_i)\}_{i \in [t]}$  (if malicious party  $P_C$  encrypts  $z$ ) to  $\mathcal{F}_{2\text{PSWHE}}$ .

It is obvious that  $\mathcal{G}_4$  and  $\mathcal{G}_5$  have the identical distribution. In hybrid  $\mathcal{G}_5$ , honest party  $P_H$  obtains a set of key-dependent parameters  $\text{keypar}'$ , which is sampled uniformly at random. In the ideal-world execution,  $\text{keypar}'$  is output by  $\mathcal{F}_{2\text{PSWHE}}$  through running  $\text{KeyGen}'$ . This is computationally indistinguishable under the RLWE assumption with KDM security. According to the analysis in  $\mathcal{G}_4$ , the ciphertexts on  $z$  obtained by honest party  $P_H$  are indistinguishable from that output by  $\mathcal{F}_{2\text{PSWHE}}$  via running the modified encryption algorithm. In conclusion,  $\mathcal{G}_5$  is computationally indistinguishable from the ideal-world execution, which completes the proof.  $\square$