2024 IEEE Symposium on Security and Privacy (SP)

Scalable Mixed-Mode MPC

Radhika Garg
Northwestern University
radhikaradhika2028@u.northwestern.edu

Kang Yang*
State Key Laboratory of Cryptology
yangk@sklc.org

Jonathan Katz*
University of Maryland
jkatz2@gmail.com

Xiao Wang*
Northwestern University
wangxiao@northwestern.edu

2024 IEEE Symposium on Security and Privacy (SP) | 979-8-3503-3130-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SP54263.2024.00106

***Abstract*—Protocols for secure multi-party computation (MPC) supporting mixed-mode computation have found a lot of applications in recent years due to their flexibility in representing the function to be evaluated. However, existing mixed-mode MPC protocols are only practical for a small number of parties: they are either tailored to the case of two/three parties, or scale poorly for a large number of parties.**

In this paper, we design and implement a new system for highly efficient and scalable mixed-mode MPC tolerating an arbitrary number of semi-honest corruptions. Our protocols allow secret data to be represented in Encrypted, Boolean, Arithmetic, or Yao form, and support efficient conversions between these representations.

1) **We design a multi-party table-lookup protocol, where both the index and the table can be kept private. The protocol is scalable even with hundreds of parties.**
2) **Using the above protocol, we design efficient conversions between additive arithmetic secret sharings and Boolean secret sharings for a large number of parties. For 32 parties, our conversion protocols require 1184× to 8141× less communication compared to the state-of-the-art protocols MOTION and MP-SPDZ; this leads to up to 1275× improvement in running time under 1 Gbps network. The improvements are even larger with more parties.**
3) **We also use new protocols to design an efficient multi-party distributed garbling protocol. The protocol could achieve asymptotically constant communication per party.**

Our implementation will be made public.

1. Introduction

Protocols for secure multi-party computation (MPC) allow a set of parties to jointly compute on their private data while revealing nothing beyond the output. In principle, general-purpose MPC protocols can evaluate an arbitrary program by first representing that program as a Boolean or arithmetic circuit; this will typically not be very efficient. *Mixed-mode* MPC protocols, on the other hand, allow different parts of a program to be represented (and securely computed) using different models of computation, e.g., part of the computation can be represented as a Boolean circuit and another part is represented using an arithmetic circuit. These protocols are of particular interest because they allow different parts of the program to be represented in the most suitable form and, therefore can achieve greater efficiency than utilizing a monolithic representation. As a result, they have found many applications, e.g., privacy-preserving machine learning and private biometric matching.

The first general-purpose MPC protocol supporting mixed-mode computation is TASTY [1], which supports conversions between garbled circuits and computation using additive homomorphic encryption in the two-party setting. It was later improved by the ABY protocol [2] that supports Boolean circuits (via garbled circuits or the GMW protocol) and arithmetic circuits (via Beaver triples). Follow up works further improve the efficiency in the two-party setting by moving some operations to offline [3], [4]. Other works have looked at decreasing the corruption threshold (e.g., [5], [6], [7]), and have shown efficiency improvements in the three-party and four-party settings with one corruption (thus honest majority). In the multi-party case, tolerating any number of corruptions, Rotaru and Wood [8] proposed mixed-mode MPC protocols supporting Boolean and arithmetic circuits for both the semi-honest and malicious settings. This was further improved in subsequent work [9], [10], [11].

Although there has been huge progress in bringing mixed-mode MPC to practical use, state-of-the-art protocols are still far from satisfactory in the following aspects:

- **Supporting MPC with massive participants.** Most existing mixed-mode MPC protocols are specifically tailored for 2–4 parties with a single corrupted party, which is useful but not sufficient. Protocols that can support an arbitrary number of parties [8], [9], [10], [11], require at least quadratic total communication complexity, rendering them inefficient for a massive number of participants.
- **Supporting high corruption thresholds.** Most protocols supporting mixed-mode computation (including all the aforementioned protocols for 2–4 parties) only allow one party to be corrupted. Exceptions are the work of Rotaru and Wood [8] and followup works [9], [10], [11] that tolerate an arbitrary number of corruptions, and MPClan [12] that assumes an honest majority.
- **Constant round complexity.** There has been a long line of work in bringing garbled circuits to the multi-party setting to reduce round complexity. However, all existing solutions require the total communication quadratic in the number of parties [13], [14], [15], [16], [17], [18]. The only exception is [19], with total communication independent of the number of parties but in the honest-majority setting.

*Corresponding authors

DOI 10.1109/SP54263.2024.00106

Note that although there exists MPC protocols for Boolean or arithmetic circuits tolerating an arbitrary number of corruptions and with communication linear in the number of parties, all existing conversion protocols (as required by mixed-mode MPC) still require quadratic communication.

1.1. Our Contribution

In this work, we design and implement a scalable MPC protocol for mixed-mode computation. We focus on the semi-honest setting with all-but-one corruptions. Our system is designed to run with a large number of parties and, crucially use encrypted representations to reduce the communication complexity in computing using each format and in converting between them. In details:

1) **Multi-party private lookup table.** We design a multi-party table-lookup protocol that takes as input a public/secret-shared table and a secret-shared index, and outputs the table value at the given index in an encrypted representation. Encrypted representation can be further switched to arithmetic secret sharings with a small cost. The protocol requires communication linear in the number of parties and is thus highly scalable.
2) **Multi-party secret-sharing conversions.** Based on the lookup-table protocol, we design efficient conversions between Boolean and arithmetic additive secret sharings. Conversion from a Boolean sharing to an arithmetic sharing is viewed as a private-index lookup in a size-2 table. Thus, the protocol has similar communication complexity as the lookup protocol and is highly scalable.
3) **Linear-complexity multi-party garbled circuits.** Based on a different variant of our table-lookup protocol, we design the first multi-party garbled-circuit protocol, tolerating an arbitrary number of semi-honest corruptions, with total communication linear in the number of parties. The protocol requires lattice-based additively homomorphic encryption in the private-key setting, and thus is not competitive with existing approaches for a small number of parties. However, we estimate that the inbound communication per party is better than quadratic-cost protocols [20], [16] for more than 128 parties.
4) **Implementation and comparison.** We propose optimizations to fully utilize the features of our protocols, and implement the protocols in a project to be open-sourced. Compared to the state-of-the-art work MOTION [11] in the same setting, for 32 parties, our system reports up to $1184\times$ improvement in communication for arithmetic-to-Boolean (A2B) conversion and $20\times$ improvement in communication for Boolean-to-arithmetic (B2A) conversion. For 64 parties, the running time of our protocols has $369\times$ improvement for A2B and $2247\times$ improvement for B2A, compared to another state-of-the-art work MP-SPDZ [21]. Note that compared to MOTION, MP-SPDZ is less efficient but supports more parties such as 64 parties under the same hardware configuration. Our protocols improve the communication cost of MP-SPDZ for 64 parties by a factor of $8819\times$ for A2B and $15384\times$ for B2A.

2. Technical Overview

Notation. We use κ and ρ to denote the computational and statistical security parameters, respectively. For a finite set S, we use $x \leftarrow S$ to denote that x is sampled uniformly from S. For a distribution $\mathcal{D}$, we denote by $x \leftarrow \mathcal{D}$ sampling x according to the distribution $\mathcal{D}$. For two integers a, b with $a \leq b$, we use $[a, b]$ to denote the set $\{a, \ldots, b\}$. We use upper-case letters like T (or bold lower-case letters like $\boldsymbol{x}$) to denote a column vector. For a vector (or bit-string) $\boldsymbol{x}$, $\boldsymbol{x}[j]$ denotes the j-th component of $\boldsymbol{x}$, where $\boldsymbol{x}[0]$ is the first component of $\boldsymbol{x}$. All arithmetic operations are computed over a finite field $\mathbb{Z}_p$, where p is a prime and $\ell = \lceil \log p \rceil$ is the length of a field element. We use $[\![x]\!]$ to denote a homomorphic encryption (HE) ciphertext on a message x, $\langle x \rangle^a$ to denote an arithmetic additive sharing over $\mathbb{Z}_p$, and $\langle x \rangle^b$ to denote a Boolean additive sharing with $x \in \{0, 1\}$. Let $P_1, \ldots, P_n$ be n parties. We use $\langle x \rangle_i^a$ or $\langle x \rangle_i^b$ to denote the share held by P_i.

2.1. Mixed-Mode MPC: Prior Solutions

Rotaru and Wood [8], who proposed doubly authenticated bits (daBits), is the first work for mixed-mode MPC in the multi-party setting tolerating any number of corruptions. Note that in the semi-honest setting, secret sharing without authentication is sufficient, but we still use daBit to refer to the underlying semi-honest construction. A daBit (in the semi-honest setting) refers to a secret bit r that is secret shared both in Boolean domain (namely $\langle r \rangle^b$) and in arithmetic domain (namely $\langle r \rangle^a$) over $\mathbb{Z}_m$ for some m. Suppose that n parties hold the secret sharing of a bit r, i.e., $\langle r \rangle^b = (r^1, \ldots, r^n)$. Party P_i, with share r^i, further secret shares the bit r^i in $\mathbb{Z}_m$ so that all parties hold $\langle r^i \rangle^a$. Now all parties need to compute $\langle r \rangle^a = \langle r^1 \rangle^a \oplus \ldots \oplus \langle r^n \rangle^a$. Note that because arithmetic sharings do not support XOR operations directly, they need to be simulated using multiplication based on the fact that $x \oplus y = x + y - 2 \cdot x \cdot y$. Because there are a total $n - 1$ number of XOR operations to compute, this protocol requires $O(n)$ multiplications over $\mathbb{Z}_m$ for each bit in the Boolean-to-arithmetic conversion. Even using multiplication triples with linear communication, the total communication for one conversion, which requires ℓ daBits, would be $O(n^2\ell^3)$ bits where ℓ is the bit length of the number to be converted.

An alternative approach by Escudero et al. [10] is to generate extended daBit (edaBit) in the form of $(\langle r \rangle^a, \langle r_0 \rangle^b, \ldots, \langle r_{\ell-1} \rangle^b)$, where $r = \sum_{j \in [0, \ell-1]} r_j \cdot 2^j \in \mathbb{Z}_m$. Their protocol works as follows: each party P_i picks a random $r^i \in \mathbb{Z}_m$ and then secretly shares r^i to all parties in both Boolean and arithmetic sharings, i.e., $\langle r^i \rangle^a$ and $\langle r^i \rangle^b$. The arithmetic sharing $\langle r \rangle^a = \sum_{i \in [1,n]} \langle r^i \rangle^a$, which can be computed for free, and Boolean sharings $(\langle r_0 \rangle^b, \ldots, \langle r_{\ell-1} \rangle^b) = \sum_{i \in [1,n]} \langle r^i \rangle^b$, which requires computing $O(n\ell)$ multi-party AND triples. Using silent OT protocols [22], [23], this requires $O(n^3\ell)$ bits of communication with a small underlying constant; or one can use threshold FHE to get $O(n^2\ell)$ where the underlying constant is at least 64, the ciphertext expansion to encrypt bits [24].

Once these (extended) daBit correlations are generated, the actual conversion can be performed easily by securely evaluating a circuit with size linear in the bit length, which is very cheap compared to the cost of generating the triples in the offline phase.

Conclusion. The above methods can be viewed as a trade-off between a larger number of parties (n) and high bit-length (ℓ). Based on these methods, follow-up works [9], [11] further optimized the concrete efficiency when $m = 2^{\ell}$, but their best variations still have a complexity of either $O(n^2\ell^3)$ or $O(n^3\ell)$.

2.2. Mixed-Mode MPC: Our Protocols

Our high-level idea is that since homomorphic encryption (HE) is a crucial tool to obtain linear communication-complexity monolithic-circuit MPC, we could also get linear communication conversions based on it. In particular, it is already known how to convert in linear complexity between additive secret sharings and the corresponding ciphertexts, with a secret key secretly shared among all parties.

Secure table lookup in the multi-party setting. Building towards scalable conversions, we first propose an efficient multi-party table lookup protocol. Suppose that we have a table T of size $m = 2^{\ell}$ containing elements in $\mathbb{Z}_p$. First, we run a cheap arithmetic sharing to encryption protocol so that P_1 holds the ciphertexts of all table entries, namely $[\![T]\!] = ([\![T[0]]\!], \ldots, [\![T[m-1]]\!])$ where $T[i]$ is the i-th table entry. If T has multiple outputs, i.e., $T[i]$ has multiple elements, then all the entries corresponding to i can be packed in a single ciphertext $[\![T[i]]\!]$. Then, P_1 picks a random string $r^1 \leftarrow \{0,1\}^{\ell}$ and locally permutes all ciphertexts to obtain $[\![T_1]\!]$ such that $T_1[j] = T[j \oplus r^1]$ for each $j \in [0, m-1]$. P_1 re-randomizes the permuted ciphertexts, and sends the resulting ciphertexts to P_2, who picks a random $r^2 \leftarrow \{0,1\}^{\ell}$ and performs a permutation to obtain $[\![T_2]\!]$ such that $T_2[j] = T_1[j \oplus r^2]$ for all $j \in [0, m-1]$. Now P_2 sends all ciphertexts $[\![T_2]\!]$ to the next party after re-randomization. Finally, P_n obtains the ciphertexts that encrypt a table permuted by all parties; in other words, P_i holds r^i as the share of r, and the ciphertexts on the permuted table $[\![T_n]\!]$ such that for each $j \in [0, m-1]$, $T_n[j] = T[j \oplus r^1 \oplus \ldots \oplus r^n] = T[j \oplus r]$. Now with this setup, a private lookup to this table on index j can be performed efficiently given a Boolean sharing $\langle j \rangle^b$: the parties compute locally and reconstruct $\langle j \oplus r \rangle^b = \langle j \rangle^b \oplus \langle r \rangle^b$ to P_n, who fetches $[\![T_n[j \oplus r]]\!] = [\![T[j]]\!]$. Then all parties convert it to an arithmetic sharing of the underlying plaintext $T[j]$.

Boolean-to-arithmetic (B2A) conversion. Our main idea for efficient conversion from Boolean to arithmetic secret sharing is to view this conversion as a lookup of a public table using a private index. In more detail, we use a public table of size 2 with 0 and 1 in $\mathbb{Z}_p$. To convert the Boolean sharing $\langle x \rangle^b$ of an integer x to its arithmetic sharing, we essentially just want to perform a table lookup for each XOR-shared bit in $\langle x \rangle^b$. This produces an arithmetic sharing of each bit in x just like daBit, which can further be locally combined to an arithmetic sharing of x.

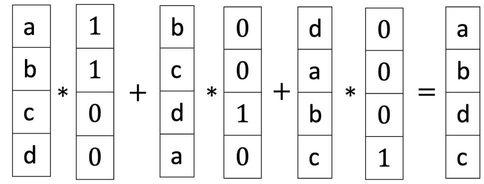


Figure 1: Example for permutation of packed table $[\![(a, b, c, d)]\!]$ using $r_1 = 0$ and $r_2 = 1$.

The idea is simple, but to make it highly efficient, extensive protocol optimization is required to incorporate state-of-the-art optimization on HE schemes. In particular, the description above does not assume packing, which is important in reducing ciphertext expansion. To make it compatible with packing, we design a customized protocol for size-2 table lookup. The main challenge is to independently permute the encrypted entries within each table that are all packed into the same ciphertext as efficiently as possible. For illustration, suppose that we have two size-2 tables, namely (a, b) and (c, d). To fully utilize packing, they will be packed in one ciphertext as $e = [\![(a, b, c, d)]\!]$. The key observation is that for a pair-wise swap, any slot after the swap can only come from its immediate neighbors, and thus shifting by one slot is sufficient. In more detail, we locally left shift and right shift e so that $e_1 = [\![(b, c, d, a)]\!]$ and $e_{-1} = [\![(d, a, b, c)]\!]$. Suppose that we use bits r_1 and r_2 to indicate whether we should swap the table entries. Then the final result is

$$(\overline{r_1}, \overline{r_1}, \overline{r_2}, \overline{r_2}) * e + (r_1, 0, r_2, 0) * e_1 + (0, r_1, 0, r_2) * e_2,$$

which can be computed with three scalar multiplications, all in one layer. We illustrate an example in Figure 1. To finish up one party's computation, it needs to re-randomize the ciphertext using the circuit-privacy technique such as noise flooding [25], [26] to ensure that r_1 and r_2 cannot be inferred from the resulting ciphertext.

Arithmetic-to-Boolean (A2B) conversion. Our protocol for arithmetic-to-Boolean conversion follows similar ideas as above but with some extra complications. Our end goal is to generate edaBit correlations over $\mathbb{Z}_p$, but the above protocol only generates edaBit correlations over $\mathbb{Z}_{2^{\ell}}$, which can be higher than p for some probability. Thus, we need a protocol to perform secure rejection sampling efficiently. The simplest way is to perform a secure comparison, but the cost would be high. Furthermore, to be compatible with the mainstream lattice-based scheme, p must be NTT-friendly, imposing more restriction. In Section 5.2, we discuss how to pick p so that comparing a private integer and p takes only 2 multiplication operations.

With the above described optimizations, our conversion protocols have a running time linear in the number of parties. We observe a significant improvement in running time and communication compared to the prior state-of-the-art work. Furthermore, we estimate the cost of several end-to-end applications using mixed-mode circuits. We observe an improvement of about 1490× in the monetary cost for running biometric matching with 64 parties.

2.3. Scalable Multi-Party Garbled Circuits

As mentioned in the introduction, all existing multi-party garbled circuit (GC) protocols in the all-but-one corruption setting require total communication quadratic in the number of parties. The closest is [15], which achieves linear online communication for the GCs (i.e., the function-dependent phase) by using a key and message homomorphic PRF, which can be built from, e.g., lattice-based assumptions. However, to distributedly generate the garbled circuits in the preprocessing phase (i.e., producing the additive sharings of keys), it still requires communication quadratic in the number of parties.

We build on the prior work [15], and achieve the total communication *linear* in the number of parties. In their protocol, each wire w is associated with two keys $\boldsymbol{k}_{w,0}$ and $\boldsymbol{k}_{w,1}$ and a random mask λ_w. These keys and masks are additively shared among all parties. Due to the key-homomorphic property, the parties can evaluate their shares of the PRF values locally and later combine them together. Ignoring some details, the protocol works as follows. For each gate g with input wires u, v and output wire w, for each $\alpha, \beta \in \{0,1\}$, each party P_i computes $\mathsf{PRF}_{\boldsymbol{k}^i_{u,\alpha}+\boldsymbol{k}^i_{v,\beta}}(g\|\alpha\|\beta) + \langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a_i$, where $e_{w,\alpha,\beta} = g((\lambda_u \oplus \alpha), (\lambda_v \oplus \beta)) \oplus \lambda_w$, $\boldsymbol{k}^i_{u,\alpha}, \boldsymbol{k}^i_{v,\beta}$ are the shares of P_i for two keys $\boldsymbol{k}_{u,\alpha}, \boldsymbol{k}_{v,\beta}$ and $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a_i$ is the arithmetic share of P_i on the key $\boldsymbol{k}_{w,e_{w,\alpha,\beta}}$. The main communication cost is to compute $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$. In [15], this was accomplished by an OT-based protocol, which requires the $O(n^2)$-communication for n parties.

Our key observation is that the computation of $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$ can be viewed as a secure table lookup. In particular, computing an arithmetic sharing $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$ boils down to computing a Boolean sharing $\langle e_{w,\alpha,\beta}\rangle^b$, which can be generated in a small communication using one random Beaver triple over binary field. We use $\langle e_{w,\alpha,\beta}\rangle^b$ to perform a table lookup, which has an efficient instantiation with $O(n)$-communication in previous discussions. For every AND gate, we need to compute 4 private table lookups, each corresponding to one arithmetic sharing $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$ with $\alpha, \beta \in \{0,1\}$. For each XOR gate, we need to compute only one private table lookup. In particular, for each XOR gate with input wires u, v and output wire w, we observe that $e_{w,\alpha,\beta} = \alpha \oplus \lambda_u \oplus \beta \oplus \lambda_v \oplus \lambda_w$, and thus one table value is $\boldsymbol{k}_{w,e_{w,0,0}} = \boldsymbol{k}_{w,e_{w,1,1}}$ and the other value in the table is $\boldsymbol{k}_{w,e_{w,1,0}} = \boldsymbol{k}_{w,e_{w,0,1}}$.

3. Preliminaries

We use the standard ideal/real paradigm [27] to prove the security of our protocols in the presence of a *semi-honest, static* adversary.

3.1. Additive Secret Sharings

We use $\langle x\rangle^t$ to denote an additive sharing over a finite field $\mathbb{F}$ in the multi-party setting, where the superscript $t \in \{a, b\}$ indicates the type of sharings. In particular, $\langle x\rangle^a$ denotes an arithmetic sharing over a field $\mathbb{F} = \mathbb{Z}_p$ where p is a prime; $\langle x\rangle^b$ represents a Boolean sharing over a field $\mathbb{F} = \mathbb{F}_2$. Then, we define the following algorithms for two types of additive sharings.

- $\langle x\rangle^t \leftarrow \mathsf{Share}(x)$: The party P_j, who holds the secret x, runs this algorithm to generate an additive sharing $\langle x\rangle^t$. Specifically, this algorithm samples $\langle x\rangle^t_i \leftarrow \mathbb{F}$ for $i \in [1, n-1]$ and computes $\langle x\rangle^t_n := x - \sum_{i\in[1,n-1]}\langle x\rangle^t_i \in \mathbb{F}$.
- $x \leftarrow \mathsf{Rec}(\langle x\rangle^t, i)$: Given all shares $\langle x\rangle^t_1, \ldots, \langle x\rangle^t_n$, P_i can run this algorithm to reconstruct the secret x. Specifically, this algorithm outputs $x := \sum_{j\in[1,n]}\langle x\rangle^t_j \in \mathbb{F}$.
- $x \leftarrow \mathsf{Open}(\langle x\rangle^t)$: The open procedure is run as follows:
  1) All parties run $\mathsf{Rec}(\langle x\rangle^t, 1)$ such that P_1 obtains x.
  2) P_1 sends x to all other parties.

It is well-known that additive secret sharings satisfy the linear property. For a vector $\boldsymbol{x} \in \mathbb{F}^\ell$, we use $\langle \boldsymbol{x}\rangle^t$ to denote $(\langle \boldsymbol{x}[0]\rangle^t, \ldots, \langle \boldsymbol{x}[\ell-1]\rangle^t)$. By $\langle \boldsymbol{x}\rangle^t_i[j]$, we denote the share of $\langle \boldsymbol{x}[j]\rangle^t$ held by the party P_i.

3.2. Threshold Homomorphic Encryption

We use threshold homomorphic encryption (THE) to encrypt messages and perform operations over ciphertexts. In most cases, we only need THE to support linear combination (including addition and scalar multiplication) and rotation operations over ciphertexts. In a few special cases (e.g., producing Beaver triples as shown in Appendix 3.3), we require THE to additionally support one multiplication operation over two ciphertexts (i.e., depth-1 THE). Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of n parties. Our protocols work in the full-threshold setting, i.e., the secret key is shared by all parties using additive secret sharing, and no party (even if $n-1$ parties collude) can recover the secret key. Let $\mathcal{M}$ be the plaintext space. Following the previous work [26], [28], [29], a THE scheme over $\mathcal{M}$ consists of the following algorithms and protocols:

- **Setup:** $pp \leftarrow \mathsf{Setup}(1^\kappa)$. On input κ, the setup algorithm outputs a set of public parameters pp, which is an implicit input to the following algorithms and protocols.
- **Key Generation:** Every party P_i generates a share of a secret key by running $\mathsf{sk}_i \leftarrow \mathsf{SecKeyGen}(pp)$. The secret key sk is identical to $\sum_{i\in[1,n]}\mathsf{sk}_i$. All parties jointly produce a public key pk by executing a multi-party key-generation protocol $\mathsf{pk} \leftarrow \prod_{\mathsf{PubKeyGen}}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
- **Encryption:** $[\![m]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$. On input a public key pk and a plaintext $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext $[\![m]\!]$.
- **Evaluation:** We consider the following operations:
  - *Linear combination* : Given ciphertexts $[\![m_1]\!], \ldots, [\![m_\ell]\!]$ and public coefficients $c_0, c_1, \ldots, c_\ell$, one can compute a ciphertext $[\![m]\!] = \sum_{i=1}^{\ell} c_i \cdot [\![m_i]\!] + c_0$ such that $m = \sum_{i=1}^{\ell} c_i \cdot m_i + c_0$.
  - *Multiplication* : Given two ciphertexts $[\![m_1]\!], [\![m_2]\!]$, any party can compute the ciphertext $[\![m_3]\!] = [\![m_1]\!] \cdot [\![m_2]\!]$ such that $m_3 = m_1 \cdot m_2$.
- **Decryption:** Given $\mathsf{sk} = \sum_{i\in[1,n]}\mathsf{sk}_i$ and a ciphertext $[\![m]\!]$, one party can run $\mathsf{Dec}_{\mathsf{sk}}([\![m]\!])$ to obtain a plaintext m. Given the secret key's shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ and a cipher-

text $[\![m]\!]$, all parties jointly execute the decryption protocol $\prod_{\mathsf{Dec}}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, [\![m]\!])$ to let some party P_i obtain m.

We require that the THE scheme satisfies the standard correctness and CPA security.

Circuit privacy. For a ciphertext $[\![m]\!] = f([\![m_1]\!], \ldots, [\![m_\ell]\!])$ that will be decrypted, it is desirable that no parties (except for the party evaluating $[\![m]\!]$ with f) could learn the secret function f, even if they hold secret key sk. This is modeled as *circuit privacy*, whose formal definition can be found in [30]. As in prior work [31], we define an algorithm $\mathsf{CP}(ct, \mathsf{pk})$, which takes as input an evaluated ciphertext ct and public key pk, and outputs a ciphertext ct' with circuit privacy. We adopt the noise-flooding technique [26] to achieve circuit privacy (see Appendix A.1 for details). More efficient technique called "divide-and-round" [31] can also be applied to our protocols.

Instantiation and packing. In the implementation, we adopt a threshold version of the BGV-HE scheme [32], which is outlined in Appendix A.1. Other full-threshold HE schemes, such as the BFV-THE scheme [33], [34], [29] can also be applied in our protocols. For the BGV-THE scheme, every ciphertext is defined over a ring $R_q = R/qR$, and any plaintext lies in a ring $R_p = R/pR$, where $R = \mathbb{Z}[X]/(X^N + 1)$ is a polynomial ring with integer coefficients modulo $X^N + 1$, N is a power-of-two integer, and $p, q \in \mathbb{N}$ are co-prime. Based on the packing technique, we can pack N plaintexts in a single ciphertext where every plaintext in $\mathbb{Z}_p$ is placed in a different slot and support parallel evaluation of plaintexts using the single instruction multiple data (SIMD) operations. Suppose that a prime $p = 1 \pmod{2N}$ is used. We can view a ring element $a \in R_p$ as a vector in $(\mathbb{Z}_p)^N$. When using the packing technique, we often use $\boldsymbol{m} \in (\mathbb{Z}_p)^N$ and $[\![\boldsymbol{m}]\!]$ to denote a vector of plaintexts and its ciphertext, unless otherwise specified. Due to the usage of packing, we need to perform the following operations to rotate or permute the plaintext slots in a single ciphertext.

- **Rotation:** Any party can run $[\![\boldsymbol{m}']\!] \leftarrow \mathsf{Rotate}_{\mathsf{pk}}([\![\boldsymbol{m}]\!], r)$ such that $\boldsymbol{m}'$ is a vector obtained by cyclically left-shifting (resp., right-shifting) the components of $\boldsymbol{m}$ by r if $r > 0$ (resp., $r < 0$).
- **Permutation:** Any party can run $[\![\boldsymbol{u}]\!] \leftarrow \mathsf{Perm}_{\mathsf{pk}}([\![\boldsymbol{m}]\!], \pi)$ such that $\boldsymbol{u} = (\boldsymbol{m}[\pi(0)], \boldsymbol{m}[\pi(1)], \ldots, \boldsymbol{m}[\pi(N-1)])$, where π is a permutation. The permutation operation can be realized by a linear combination of multiple rotations.

It is well known that the BGV-THE scheme is CPA secure under the ring-LWE assumption [35].

3.3. Arithmetic Black Box and Conversions

We model MPC via the arithmetic black-box (ABB) model [36], which is an ideal functionality $\mathcal{F}_{\mathsf{ABB}}$ defined in Figure 2. This functionality allows a set of n parties to input/output secret-shared values and evaluate arbitrary circuits performing addition and multiplication operations. As in [10], we define an extended version of the ABB model, which handles values in both arithmetic and Boolean domains and thus can evaluate any arithmetic/boolean circuits.

Functionality $\mathcal{F}_{\mathsf{ABB}}$

This functionality operates over a finite field $\mathbb{Z}_p$ (resp., $\mathbb{F}_2$) for arithmetic secret-shared values (resp., Boolean secret-shared values), and interacts with parties $P_1, \ldots, P_n$.

Input: Upon receiving $(\mathsf{Input}, P_i, \mathsf{type}, \mathsf{id}, x)$ from a party P_i and $(\mathsf{Input}, P_i, \mathsf{type}, \mathsf{id})$ from all other parties, where $\mathsf{type} \in \{\mathsf{arith}, \mathsf{bool}\}$, id is a fresh identifier, and either $x \in \mathbb{Z}_p$ or $x \in \{0,1\}$ depending on type, store $(\mathsf{id}, \mathsf{type}, x)$.

Random: Upon receiving $(\mathsf{Random}, \mathsf{type}, \mathsf{id})$ from all parties where $\mathsf{type} \in \{\mathsf{arith}, \mathsf{bool}\}$ and id is a fresh identifier, sample $r \leftarrow \mathbb{Z}_p$ or $r \leftarrow \{0,1\}$ relying on type, store $(\mathsf{id}, \mathsf{type}, r)$.

Encrypt: Upon receiving $(\mathsf{Enc}, \mathsf{id}, \mathsf{id}')$ from all parties where id is present in memory, retrieve $(\mathsf{id}, \mathsf{type}, x)$ and store $(\mathsf{id}', \mathsf{enc}, x)$.

Linear combination: Upon receiving $(\mathsf{LinComb}, \mathsf{type}, \mathbf{id}, \mathsf{id}', c_0, c_1, \ldots, c_\ell)$ from all parties, where $(\mathbf{id}[j], \mathsf{type})$ for $j \in [1, \ell]$ are present in memory, and $c_j \in \mathbb{Z}_p$ (resp., $c_j \in \{0,1\}$) for $j \in [0, \ell]$ if $\mathsf{type} \in \{\mathsf{arith}, \mathsf{enc}\}$ (resp., $\mathsf{type} = \mathsf{bool}$), retrieve $(\mathbf{id}[j], \mathsf{type}, x_j)$ for $j \in [1, \ell]$, then compute $y := \sum_{j=1}^{\ell} c_j \cdot x_j + c_0$ modulo p if $\mathsf{type} \in \{\mathsf{arith}, \mathsf{enc}\}$ and modulo 2 if $\mathsf{type} = \mathsf{bool}$. Store $(\mathsf{id}', \mathsf{type}, y)$.

Multiply: Upon receiving $(\mathsf{Mult}, \mathsf{type}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties where $(\mathsf{id}_1, \mathsf{type})$ and $(\mathsf{id}_2, \mathsf{type})$ are present in memory and $\mathsf{type} \in \{\mathsf{arith}, \mathsf{bool}\}$, retrieve $(\mathsf{id}_1, \mathsf{type}, x)$ and $(\mathsf{id}_2, \mathsf{type}, y)$, compute $z := x \cdot y$ modulo p if $\mathsf{type} = \mathsf{arith}$ and modulo 2 if $\mathsf{type} = \mathsf{bool}$, and store $(\mathsf{id}_3, \mathsf{type}, z)$.

Output: Upon receiving $(\mathsf{Output}, P_i, \mathsf{type}, \mathsf{id})$ from all parties, where $(\mathsf{id}, \mathsf{type})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{type}, x)$ and then output it to P_i.

Figure 2: Functionality for the MPC black box.

Furthermore, this functionality is also extended to allow the parties to encrypt values and evaluate the addition of two ciphertexts. Without loss of generality, suppose that the plaintext space for encryption is $\mathbb{Z}_p$. Here, we do not allow the parties to evaluate the multiplication of two ciphertexts, as our protocols do not require it when invoking functionality $\mathcal{F}_{\mathsf{ABB}}$. This functionality abstracts away the underlying details of secret sharings, encryption, and MPC.

Instantiation for $\mathcal{F}_{\mathsf{ABB}}$. We can use a THE scheme to encrypt values and perform a linear combination of encrypted values. We adopt additive secret sharings to securely compute the linear combination and multiplication of secret values. Due to the linear property of additive secret sharings, the linear combination of multiple sharings can be locally computed. For multiplication of two secret sharings, we consider two cases:

- *Arithmetic sharings* : We can compute the multiplication of two arithmetic sharings $\langle x \rangle^a$ and $\langle y \rangle^a$ using threshold HE that supports 1-depth multiplications. Specifically, every party P_i with $i \neq 1$ runs $[\![\langle x \rangle_i^a]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\langle x \rangle_i^a)$ and $[\![\langle y \rangle_i^a]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\langle y \rangle_i^a)$, and then sends $([\![\langle x \rangle_i^a]\!], [\![\langle y \rangle_i^a]\!])$ to P_1 who computes $[\![x]\!] := \sum_{i=1}^{n} [\![\langle x \rangle_i^a]\!]$ and $[\![y]\!] := \sum_{i=1}^{n} [\![\langle y \rangle_i^a]\!]$, where $[\![x]\!]_1, [\![y]\!]_1$ are computed by P_1 by running $\mathsf{Enc}_{\mathsf{pk}}(\cdot)$. Then, P_1 locally computes $[\![z]\!] := [\![x]\!] \cdot [\![y]\!]$ and sends $[\![z]\!]$ to all other parties. Finally, all parties call the E2A command of $\mathcal{F}_{\mathsf{Conv}}$ to convert $[\![z]\!]$ into $\langle z \rangle^a$. This

Functionality $\mathcal{F}_{\mathsf{Conv}}$

This functionality has all of the same features as $\mathcal{F}_{\mathsf{ABB}}$ with the following additional commands.

From Boolean to Arithmetic: Upon receiving $(\mathsf{B2A}, \mathsf{id}, \mathsf{id}')$ from all parties where $(\mathsf{id}, \mathsf{bool})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{bool}, x)$ and store $(\mathsf{id}', \mathsf{arith}, x)$.

From Arithmetic to Boolean: Upon receiving $(\mathsf{A2B}, \mathsf{id}, \mathsf{id}_0, \ldots, \mathsf{id}_{\ell-1})$ from all parties where $(\mathsf{id}, \mathsf{arith})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{arith}, x)$ and store $(\mathsf{id}_i, \mathsf{bool}, x_i)$ for $i \in [0, \ell-1]$ where $x = \sum_{i=0}^{\ell-1} x_i \cdot 2^i \mod p$.

From Encryption to Arithmetic: Upon receiving $(\mathsf{E2A}, \mathsf{id}, \mathsf{id}')$ from all parties where $(\mathsf{id}, \mathsf{enc})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{enc}, x)$ and store $(\mathsf{id}', \mathsf{arith}, x)$.

From Arithmetic to Encryption: Upon receiving $(\mathsf{A2E}, \mathsf{id}, \mathsf{id}')$ from all parties where $(\mathsf{id}, \mathsf{arith})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{arith}, x)$ and store $(\mathsf{id}', \mathsf{enc}, x)$.

Figure 3: Functionality for the black box of conversions.

needs to send at most $3.5(n-1)$ HE ciphertexts in total, when the E2A command of $\mathcal{F}_{\mathsf{Conv}}$ is instantiated by the protocol shown in Figure 14 of Section A.3.

- *Boolean sharings* : We can use the above approach based on threshold HE to multiply two Boolean sharings $\langle x\rangle^b, \langle y\rangle^b$, where XOR is simulated by multiplication and addition operations. In this way, the communication complexity of $O(n)$ can be achieved, but the computation complexity is high. Alternatively, we can adopt the standard protocol based on correlated oblivious transfer (COT) to perform pairwise bit multiplications, and then locally combine the shares of these bit multiplications to obtain Boolean sharing $\langle z\rangle^b$ with $z = x \wedge y$. We can use the recent PCG-like COT protocols (e.g., [22], [23]) to generate COT correlations. Although the COT approach has the communication complexity of $O(n^2)$, it allows us to obtain much faster computation.

We can adopt the Beaver's multiplication technique to improve the online performance. In this case, the online communication per multiplication is $4(n-1)\log p$ bits (resp., $4(n-1)$ bits) in total for $t = a$ (resp., $t = b$).

Functionality for arithmetic, Boolean and encryption conversions. Our protocol would securely realize functionality $\mathcal{F}_{\mathsf{Conv}}$ shown in Figure 3. This functionality allows the parties to convert between arithmetic secret-shared values and Boolean secret-shared values and also allows them to convert between arithmetic secret-shared values and encrypted values. We omit the conversions between Boolean secret-shared values and encrypted values, as they can be realized by performing Boolean-to-arithmetic and arithmetic-to-encryption conversions. As for the conversions between arithmetic secret-shared values and encrypted values, we w.l.o.g. assume that the space of secret values for arithmetic sharings is the same as that of HE plaintexts. We show how to perform efficient conversions between arithmetic sharings and encrypted values in Appendix A.3 and will present our conversion protocols between arithmetic sharings and Boolean sharings in Section 5.

Functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$

Let $M = 2^m$ be the length of a public/private table. This functionality has all of the same features as $\mathcal{F}_{\mathsf{ABB}}$ shown in Figure 2, with the following additional commands.

Public masked table: Upon receiving $(\mathsf{MaskedPubTab}, T, \mathbf{id}_1, \mathbf{id}_2)$ from all parties, where $T \in (\mathbb{Z}_p)^M$ is a vector defining a public table, and $\mathbf{id}_1, \mathbf{id}_2$ are two vectors of fresh identifiers with respective length m and M, sample $r \leftarrow \{0,1\}^m$, write $r = (r_0, \ldots, r_{m-1})$ with $r_j \in \{0,1\}$ for $j \in [0, m-1]$, store $(\mathbf{id}_1[j], \mathsf{bool}, r_j)$ for $j \in [0, m-1]$, and store $(\mathbf{id}_2[j], \mathsf{arith}, T[r \oplus j])$ for $j \in [0, M-1]$.

Private masked table: Upon receiving $(\mathsf{MaskedPriTab}, \mathbf{id}_1, \mathbf{id}_2, \mathbf{id}_3)$ from all parties, where $(\mathbf{id}_1[j], \mathsf{arith})$ for all $j \in [0, M-1]$ are present in memory, and $\mathbf{id}_2, \mathbf{id}_3$ are two vectors of fresh identifiers with respective length m and M, retrieve $(\mathbf{id}_1[j], \mathsf{arith}, T[j])$ for $j \in [0, M-1]$, set the vector T accordingly, sample $r \leftarrow \{0,1\}^m$ with $r = (r_0, \ldots, r_{m-1})$, store $(\mathbf{id}_2[j], \mathsf{bool}, r_j)$ for $j \in [0, m-1]$, and store $(\mathbf{id}_3[j], \mathsf{arith}, T[r \oplus j])$ for $j \in [0, M-1]$.

Figure 4: Functionality for masked lookup tables.

Functionality $\mathcal{F}_{\mathsf{LUT}}$

Let $M = 2^m$ be the length of a public/private table. This functionality has all of the same features as $\mathcal{F}_{\mathsf{ABB}}$ shown in Figure 2, with the following additional commands.

Public table lookup: Upon receiving $(\mathsf{PubTabLookup}, T, \mathbf{id}_1, \mathsf{id}_2)$ from all parties, where $T \in (\mathbb{Z}_p)^M$ is a vector defining a public table, and $(\mathbf{id}_1[j], \mathsf{bool})$ for $j \in [0, m-1]$ are present in memory, retrieve $(\mathbf{id}_1[j], \mathsf{bool}, x[j])$ for $j \in [0, m-1]$, set $x \in \{0,1\}^m$ and store $(\mathsf{id}_2, \mathsf{arith}, T[x])$.

Private table lookup: Upon receiving $(\mathsf{PriTabLookup}, \mathbf{id}_1, \mathbf{id}_2, \mathsf{id}_3)$ from all parties, where $(\mathbf{id}_1[j], \mathsf{arith})$ for all $j \in [0, M-1]$ and $(\mathbf{id}_2[j], \mathsf{bool})$ for $j \in [0, m-1]$ are present in memory, retrieve $(\mathbf{id}_1[j], \mathsf{arith}, T[j])$ for $j \in [0, M-1]$ and $(\mathbf{id}_2[j], \mathsf{bool}, x[j])$ for $j \in [0, m-1]$, set $T \in (\mathbb{Z}_p)^m$ and $x \in \{0,1\}^m$ accordingly, and store $(\mathsf{id}_3, \mathsf{arith}, T[x])$.

Figure 5: Functionality for MPC using look-up tables.

4. Multi-Party Lookup-Table Protocol

In this section, we present a multi-party lookup-table protocol with linear communication complexity, where either the table is public, or a private table is secretly shared. We separate the lookup-table protocol into two sub-protocols, where the preprocessing sub-protocol generates a masked table and the online sub-protocol realizes the lookup table using the masked table. We model the preprocessing of public/private masked tables in functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$ shown in Figure 4. Functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$ samples a random string r to permute the table T, which is equivalent to generating the additive sharings of masked table T' and r, where $T'[j] = T[r \oplus j]$ for $j \in [0, M-1]$. By invoking $\mathcal{F}_{\mathsf{Prep-LUT}}$, our online protocol securely realizes functionality $\mathcal{F}_{\mathsf{LUT}}$ shown in Figure 5. Both functionalities $\mathcal{F}_{\mathsf{Prep-LUT}}$ and $\mathcal{F}_{\mathsf{LUT}}$ are an extension of the lookup-table functionality [37] to additionally support private tables. In both $\mathcal{F}_{\mathsf{Prep-LUT}}$ and $\mathcal{F}_{\mathsf{LUT}}$, we w.l.o.g. assume that the table

Protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$

Inputs: Parties $P_1, \ldots, P_n$ have the following inputs:

- *Case 1* : Let $T \in (\mathbb{Z}_p)^2$ be a public vector corresponding to a size-2 public table. *Case 2* : Let $\langle T \rangle^a$ be an arithmetic sharing w.r.t. a private vector $T \in (\mathbb{Z}_p)^2$ that is related to a size-2 private table.
- The THE scheme with Enc and Rotate. Suppose that the public parameters and public key pk have been established.

Preprocessing of a masked size-2 table:

1) All parties call the (Random) command of functionality $\mathcal{F}_{\mathsf{E2A}}$ to sample a vector of random Boolean sharings $\langle r \rangle^b$ with $r = (r_0, r_1, \ldots, r_{N/2-1}) \in \{0,1\}^{N/2}$.
2) P_1 obtains a ciphertext $[\![\boldsymbol{m}]\!]$ where $\boldsymbol{m}[2j] = T[0]$ and $\boldsymbol{m}[2j+1] = T[1]$ for $j \in [0, N/2-1]$, by executing the following depending on whether T is public or not.
   - In Case 1, P_1 sets $\boldsymbol{m}$ as above and runs $[\![\boldsymbol{m}]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m})$.
   - In Case 2, every party P_i sets $\boldsymbol{m}_i \in (\mathbb{Z}_p)^N$ as $\boldsymbol{m}[2j] = \langle T \rangle^a_i[0]$ and $\boldsymbol{m}[2j+1] = \langle T \rangle^a_i[1]$ for $j \in [0, N/2-1]$, and then runs $[\![\boldsymbol{m}_i]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m}_i)$. For each $i \neq 1$, P_i sends $[\![\boldsymbol{m}_i]\!]$ to P_1, who computes $[\![\boldsymbol{m}]\!] := \sum_{i\in[1,n]} [\![\boldsymbol{m}_i]\!]$.
3) From $i = 1$ to n, the parties execute the following steps:
   a) P_1 sets $c_0 := [\![\boldsymbol{m}]\!]$. If $i \neq 1$, P_i gets c_{i-1} from P_{i-1}.
   b) P_i computes two ciphertexts $t_1 \leftarrow \mathsf{Rotate}_{\mathsf{pk}}(c_{i-1}, 1)$ and $t_2 \leftarrow \mathsf{Rotate}_{\mathsf{pk}}(c_{i-1}, -1)$.
   c) P_i initializes three zero-vectors $\boldsymbol{h} = \boldsymbol{h}_1 = \boldsymbol{h}_2 = 0^N$, and for each $j \in [0, N/2-1]$, does the following:
      i) If $\langle r_j \rangle^b_i = 0$, then set $\boldsymbol{h}[2j] = \boldsymbol{h}[2j+1] = 1$.
      ii) If $\langle r_j \rangle^b_i = 1$, then set $\boldsymbol{h}_1[2j] = \boldsymbol{h}_2[2j+1] = 1$.
   d) P_i computes $c_i := \boldsymbol{h} \cdot c_{i-1} + \boldsymbol{h}_1 \cdot t_1 + \boldsymbol{h}_2 \cdot t_2$, and then update c_i as a circuit-private ciphertext $\mathsf{CP}(c_i, \mathsf{pk})$.
   e) If $i \neq n$, P_i sends c_i to P_{i+1}. If $i = n$, P_n sends c_n to all parties.
4) The parties call functionality $\mathcal{F}_{\mathsf{E2A}}$ to convert ciphertext c_n into arithmetic sharings $\langle T'_i \rangle^a$ for $i \in [0, N/2-1]$, where $T'_i[j] = T[r_i \oplus j]$ for $i \in [0, N/2-1]$, $j \in \{0, 1\}$.
5) The parties output additive sharings $\langle r_i \rangle^b$ and $\langle T'_i \rangle^a$ for $i \in [0, N/2-1]$.

Figure 6: Protocol for the preprocessing of masked size-2 tables in the $\mathcal{F}_{\mathsf{E2A}}$-hybrid model.

size is power-of-two. As in [37], we also let $\mathcal{F}_{\mathsf{Prep-LUT}}$ and $\mathcal{F}_{\mathsf{LUT}}$ involve the commands defined in functionality $\mathcal{F}_{\mathsf{ABB}}$ shown in Figure 2. In Section 4.1 and Section 4.2, we describe two preprocessing protocols for generating masked tables. Then, we present the online protocol in Section 4.3.

Our conversion and multi-party garbling protocols shown in the next sections only need a lookup-table protocol for *size-2* tables. Therefore, we first describe the multi-party lookup-table protocol for size-2 tables. Then, we show how to construct a multi-party lookup-table protocol for any polynomial-sized tables, which may be of independent interest for other applications, e.g., secure AES evaluation.

4.1. Preprocessing for Masked Two-Sized Tables

In Figure 6, we show a multi-party preprocessing protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ for masking the tables that have only two entries. This protocol works in the $\mathcal{F}_{\mathsf{E2A}}$-hybrid model, where $\mathcal{F}_{\mathsf{E2A}}$ consists of the commands defined in $\mathcal{F}_{\mathsf{ABB}}$ (shown in Figure 2) and the E2A command defined in $\mathcal{F}_{\mathsf{Conv}}$ (shown in Figure 3). Besides, this protocol adopts a THE scheme supporting packing and SIMD, where the THE scheme adopts the plaintext space $(\mathbb{Z}_p)^N$ for a prime p, and the number of plaintext slots is N (power of two).

Theorem 1. *Protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ (shown in Figure 6) securely realizes functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$ with size-2 tables (shown in Figure 4) against semi-honest adversaries in the $\mathcal{F}_{\mathsf{E2A}}$-hybrid model, assuming that the THE scheme is CPA secure and satisfies circuit privacy.*

The proof of Theorem 1 is provided in Appendix B.1.

Reducing noise growth. Every ciphertext c_i produced by P_i for $i \geq 2$ is computed by performing rotation and scalar-multiplication operations over ciphertext c_{i-1}. In this case, the noise will grow with the number of parties, which will lead to very large parameters. To solve the issue, we reduce the noise growth by refreshing every ciphertext c_i with the Bootstrapping operation. This optimization is described in Section 7.1, and keeps the ciphertext size almost constant. Instead of bootstrapping every ciphertext, one only needs to bootstrap the ciphertexts $\{c_j\}$ where $j = i \cdot k$ for $i \in [1, n/k]$ and some integer $k \geq 2$, by tuning the parameters.

Special case that table entries are in $(\mathbb{Z}_p)^N$. When each table entry is taken from the message space of the packed THE scheme (i.e., $T[0], T[1] \in (\mathbb{Z}_p)^N$), we can use a simpler approach to generate a masked lookup table. The special case occurs in the application of our multi-party garbling protocol shown in Section 6. The preprocessing protocol for the special case is the same as the protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ shown in Figure 6, except for the following differences:

1) A random Boolean sharing $\langle r \rangle^b$ with $r \in \{0, 1\}$ (instead of $r \in \{0,1\}^{N/2}$) is generated.
2) A THE ciphertext on a public/private table T is computed as $([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!])$, where $\boldsymbol{x} = T[0]$ and $\boldsymbol{y} = T[1]$.
3) From $i = 1$ to n, P_i does the following:
   a) If $i = 1$, set $c_0 = ([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!])$. If $i \neq 1$, receive c_{i-1} from P_{i-1}.
   b) Parse ciphertext $c_{i-1} = ([\![\boldsymbol{x}_{i-1}]\!], [\![\boldsymbol{y}_{i-1}]\!])$. If $\langle r \rangle^b_i = 1$, then *swap* $([\![\boldsymbol{x}_{i-1}]\!], [\![\boldsymbol{y}_{i-1}]\!])$ and update c_{i-1} accordingly. Otherwise, keep c_{i-1} unchanged.
   c) Compute THE ciphertexts with circuit privacy $c_i := (\mathsf{CP}(c_{i-1}[0], \mathsf{pk}), \mathsf{CP}(c_{i-1}[1], \mathsf{pk}))$.

Through the above approach, only one table (instead of $N/2$ tables) is masked for each protocol execution. In the special case, it is unnecessary to bootstrap ciphertexts, as only circuit-privacy operations are involved and noise growth is slower. Therefore, for the special case, this protocol is more computation-efficient than the protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$.

4.2. Preprocessing for Masked Poly-Sized Tables

Now, we describe the multi-party preprocessing protocol for masking any polynomial-sized tables. This protocol still works in the $\mathcal{F}_{\mathsf{E2A}}$-hybrid model, and adopts the threshold

Protocol $\Pi_{\mathsf{prepLUT}}^{\mathsf{polysize}}$

Inputs: Parties $P_1, \ldots, P_n$ have the following inputs:

- *Case 1* : Let $T \in (\mathbb{Z}_p)^M$ be a vector corresponding to a size-M public table. *Case 2* : Let $\langle T \rangle^a$ be an arithmetic sharing w.r.t. a vector $T \in (\mathbb{Z}_p)^M$ related to a size-M private table. Let m be the length of indices, i.e., $M = 2^m$.
- The THE scheme with Enc and Perm. Suppose that the public parameters and public key pk have been established.

Preprocessing of a masked size-M table:

1) All parties call the (Random) command of $\mathcal{F}_{\mathsf{E2A}}$ to sample random Boolean sharings $\langle r \rangle^b$ with $r \in \{0,1\}^m$.
2) P_1 computes a ciphertext $[\![T]\!]$ by executing the following depending on if T is public.
   - In Case 1, P_1 runs $[\![T]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(T)$.
   - In Case 2, every party P_i runs $[\![T^i]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\langle T \rangle_i^a)$. For $i \neq 1$, P_i sends $[\![T^i]\!]$ to P_1, who computes $[\![T]\!] := \sum_{i \in [1,n]} [\![T^i]\!]$.
3) From $i = 1$ to n, the parties execute the following steps:
   a) If $i = 1$, P_1 sets $[\![T_0]\!] := [\![T]\!]$. If $i \neq 1$, P_i receives $[\![T_{i-1}]\!]$ from P_{i-1}.
   b) P_i defines π_i as $\pi_i(j) = j \oplus \langle r \rangle_i^b \in \{0,1\}^m$ for $j \in [0, M-1]$, and then runs $[\![T_i]\!] \leftarrow \mathsf{Perm}_{\mathsf{pk}}([\![T_{i-1}]\!], \pi_i)$. Then P_i updates $[\![T_i]\!]$ as a circuit-private ciphertext $\mathsf{CP}([\![T_i]\!], \mathsf{pk})$.
   c) If $i \neq n$, P_i sends $[\![T_i]\!]$ to P_{i+1}. If $i = n$, P_n sends $[\![T_n]\!]$ to all other parties.
4) The parties call $\mathcal{F}_{\mathsf{E2A}}$ to convert ciphertext $[\![T_n]\!]$ into a vector of arithmetic sharings $\langle T' \rangle^a$ with $T'[j] = T[r \oplus j]$ for $j \in [0, m-1]$.
5) The parties output additive sharings $\langle r \rangle^b$ and $\langle T' \rangle^a$.

Figure 7: Protocol for the preprocessing of masked poly-sized tables in the $\mathcal{F}_{\mathsf{E2A}}$-hybrid model.

HE scheme to encrypt the public/private table. This protocol makes the parties sequentially permute the encrypted table, and requires more rotation operations compared to the protocol $\Pi_{\mathsf{prepLUT}}^{\mathsf{size2}}$ shown in Figure 6. The details of the protocol is shown in Figure 7. Similarly, to control the noise growth, we would adopt the bootstrapping technique to refresh evaluated ciphertexts. For the sake of simplicity, we do not involve the packing technique for the THE scheme. Below, we will give the overview how to adopt the packing technique to optimize the protocol for moderate-sized tables.

Theorem 2. *Protocol $\Pi_{\mathsf{prepLUT}}^{\mathsf{polysize}}$ (shown in Figure 7) securely realizes functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$ with poly-sized tables (shown in Figure 4) against semi-honest adversaries in the $\mathcal{F}_{\mathsf{E2A}}$-hybrid model, assuming that the THE scheme is CPA secure and satisfies circuit privacy.*

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

Optimization with packing. When a packed THE scheme is adopted, we can further optimize the protocol if the table size M is two times smaller than the number of slots N. Let $L = \lfloor N/M \rfloor$. In this case, we can encrypt and permute L tables packed in a single ciphertext. In particular, every party can permute each encrypted table independently and randomly. For a moderate table size M, we can select a suitable parameter N to obtain a better efficiency.

Protocol Π_{Lookup}

Inputs: Parties $P_1, \ldots, P_n$ have the following inputs:

- Let $M = 2^m$ is the table length.
- *Case 1* : Let $T \in (\mathbb{Z}_p)^M$ be a public vector corresponding to a public-table map $f : \{0,1\}^m \to \mathbb{Z}_p$ such that $T[j] = f(j)$ for $j \in \{0,1\}^m$. *Case 2* : Let $\langle T \rangle^a$ be the arithmetic sharing of a private vector T related to a private-table map.
- $\langle x \rangle^b$ is the Boolean sharings of a private index $x \in \{0,1\}^m$.

Preprocessing of masked table: In Case 1 (resp., Case 2), all parties call the $(\mathsf{MaskedPubTab}, T)$ (resp., $(\mathsf{MaskedPriTab})$) command of functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$ to generate a masked table $(\langle r \rangle^b, \langle T' \rangle^a)$ with $r \in \{0,1\}^m$ and $T'[j] = T[r \oplus j]$ for $j \in [0, M-1]$.

Online table lookup: Given $(\langle r \rangle^b, \langle T' \rangle^a)$ and $\langle x \rangle^b$, the parties generate $\langle T[x] \rangle^a$ as follows:

1) All parties locally compute $\langle u \rangle^b := \langle x \rangle^b \oplus \langle r \rangle^b$.
2) The parties run the $\mathsf{Open}(\langle u \rangle^b)$ procedure such that they obtain $u = x \oplus r \in \{0,1\}^m$.
3) The parties locally compute $\langle T[x] \rangle^a := \langle T' \rangle^a[u]$.

Figure 8: Online lookup-table protocol.

4.3. Online Protocol for Lookup Table

The online lookup-table protocol follows the known approach [38], [39], [37], [40], [41], and allows the table to be public or private. The detailed protocol is shown in Figure 8, and works in the $\mathcal{F}_{\mathsf{Prep-LUT}}$-hybrid model. This protocol takes an input a vector of Boolean sharings $\langle x \rangle^b$ with $x \in \{0,1\}^m$ and outputs an arithmetic sharing $\langle T[x] \rangle^a$.

Theorem 3. *Protocol Π_{Lookup} (shown in Figure 8) securely realizes functionality $\mathcal{F}_{\mathsf{LUT}}$ in the presence of semi-honest adversaries in the $\mathcal{F}_{\mathsf{Prep-LUT}}$-hybrid model.*

The proof of Theorem 3 is postponed to Appendix B.3.

5. Conversions of Sharings from LUT

We first show how to convert Boolean sharings into arithmetic sharings in the $\mathcal{F}_{\mathsf{LUT}}$-hybrid model. Then, we describe the protocol to convert arithmetic sharings into Boolean sharings in the $\mathcal{F}_{\mathsf{LUT}}$-hybrid model.

5.1. Boolean to Arithmetic Conversion

In Figure 9, we show a LUT-based protocol that converts a vector of Boolean sharings $\langle \boldsymbol{x} \rangle^b$ into an arithmetic sharing $\langle x \rangle^a$ with $x = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{x}[j] \mod p$. Specifically, all parties compute an arithmetic sharing for each bit $\boldsymbol{x}[j]$ by defining a public table $T = (0,1)$ and calling $\mathcal{F}_{\mathsf{LUT}}$. Then, the parties locally sum the arithmetic sharings on all bits to get $\langle x \rangle^a$.

Theorem 4. *Protocol Π_{B2A} (shown in Figure 9) securely realizes the $\mathsf{B2A}$ command of functionality $\mathcal{F}_{\mathsf{Conv}}$ against semi-honest adversaries in the $\mathcal{F}_{\mathsf{LUT}}$-hybrid model.*

The proof of Theorem 4 is given in the full version [42].

Protocol Π_{B2A}

Inputs: $P_1, \ldots, P_n$ hold Boolean sharings $\langle \boldsymbol{x} \rangle^b$ where $\boldsymbol{x} \in (\mathbb{F}_2)^\ell$ and $\ell = \lceil \log p \rceil$ is the length of an element in $\mathbb{Z}_p$.

Conversion from Boolean to arithmetic sharings:

1) All parties set a pubic vector $T = (0, 1)$ corresponding to a public lookup-table $f_T(j) = j$ for $j \in \{0, 1\}$.
2) For $j \in [0, \ell - 1]$, the parties call the $(\mathsf{PubTabLookup})$ command of functionality $\mathcal{F}_{\mathsf{LUT}}$ on input $(T, \langle \boldsymbol{x}[j] \rangle^b)$ to obtain $\langle T[\boldsymbol{x}[j]] \rangle^a = \langle \boldsymbol{x}[j] \rangle^a$.
3) The parties compute and output $\langle x \rangle^a := \sum_{j=0}^{\ell-1} 2^j \cdot \langle \boldsymbol{x}[j] \rangle^a$, where $x = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{x}[j] \mod p$.

Figure 9: Protocol for converting Boolean sharings to arithmetic sharings in the $\mathcal{F}_{\mathsf{LUT}}$-hybrid model.

Protocol Π_{A2B}

Inputs: Parties $P_1, \ldots, P_n$ hold an arithmetic sharing $\langle x \rangle^a$ with $x \in \mathbb{Z}_p$. Let $\ell = \lceil \log p \rceil$.

Conversion from arithmetic to Boolean sharings:

1) All parties call the (Random) command of $\mathcal{F}_{\mathsf{LUT}}$ to sample a vector of Boolean sharings $\langle \boldsymbol{r} \rangle^b$ with $\boldsymbol{r} \in (\mathbb{F}_2)^\ell$.
2) The parties execute Π_{B2A} shown in Figure 9 to obtain an arithmetic sharing $\langle r \rangle^a$ with $r = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{r}[j]$.
3) If $(2^\ell - p)/2^\ell > 1/2^\rho$, all parties call the $\mathsf{LinComb}$ and Mult commands of $\mathcal{F}_{\mathsf{LUT}}$ on input $\langle \boldsymbol{r} \rangle^b$ to check if $r = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{r}[j] < p$. If $r \geq p$, the parties go back to step 1.
4) All parties locally compute $\langle u \rangle^a := \langle x \rangle^a - \langle r \rangle^a$. Then, the parties run $\mathsf{Rec}(\langle u \rangle^a, 1)$ to make P_1 reconstruct $u = (x - r) \mod p$, and locally define a vector of Boolean sharings $\langle \boldsymbol{u} \rangle^b$ via letting P_1 set $\langle \boldsymbol{u} \rangle_1^b[j]$ as the j-th bit of the bit-decomposition of $u \in \mathbb{Z}_p$ and letting P_i for $i \neq 1$ set $\langle \boldsymbol{u} \rangle_i^b[j] = 0$ for $j \in [0, \ell - 1]$.
5) The parties call the Input, $\mathsf{LinComb}$ and Mult commands of $\mathcal{F}_{\mathsf{LUT}}$ on input $(\langle \boldsymbol{u} \rangle^b, \langle \boldsymbol{r} \rangle^b)$ to compute a modulo-addition circuit, which takes as input $\boldsymbol{u}, \boldsymbol{r} \in \{0,1\}^\ell$ and outputs $u + r \mod p$. Functionality $\mathcal{F}_{\mathsf{LUT}}$ returns $\langle \boldsymbol{x} \rangle^b$ to the parties, where $x = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{x}[j] \mod p$.
6) The parties output $\langle \boldsymbol{x} \rangle^b$ with $\boldsymbol{x} \in (\mathbb{F}_2)^\ell$.

Figure 10: Protocol for converting arithmetic sharings to Boolean sharings in the $\mathcal{F}_{\mathsf{LUT}}$-hybrid model.

5.2. Arithmetic to Boolean Conversion

In Figure 10, we describe a protocol to convert an arithmetic sharing $\langle x \rangle^a$ to Boolean sharings $\langle \boldsymbol{x} \rangle^b$ with $x = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{x}[j] \mod p$. This protocol also works in the $\mathcal{F}_{\mathsf{LUT}}$-hybrid model where $\mathcal{F}_{\mathsf{LUT}}$ involves the commands defined in $\mathcal{F}_{\mathsf{ABB}}$ (shown in Figure 2), and invokes Π_{B2A} (shown in Figure 9) as a sub-protocol. Specifically, the parties sample a vector of random Boolean sharings $\langle \boldsymbol{r} \rangle^b$ and run Π_{B2A} to get $\langle r \rangle^a$, where $\boldsymbol{r} \in (\mathbb{F}_2)^\ell$ and $r = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{r}[j] \mod p$. They open $u = x - r$, and jointly compute a modulo-addition circuit with input $\langle \boldsymbol{u} \rangle^b$ and $\langle \boldsymbol{r} \rangle^b$ to obtain $\langle \boldsymbol{x} \rangle^b$, where $\langle \boldsymbol{u} \rangle^b$ can be locally computed by the parties given u.

If $(2^\ell - p)/2^\ell \leq 1/2^\rho$, then $r = \sum_{j=0}^{\ell-1} 2^j \cdot \boldsymbol{r}[j] \mod p$ is indistinguishable from a uniform element in $\mathbb{Z}_p$ except with probability at most $1/2^\rho$, where $\boldsymbol{r}$ is a random vector in $(\mathbb{F}_2)^\ell$. Otherwise, we need a check if $r < p$ to assure that r is random in $\mathbb{Z}_p$ and re-sample $\langle \boldsymbol{r} \rangle^b$ if $r \geq p$. In general, the parties compute a comparison circuit with input $\langle \boldsymbol{r} \rangle^b$ to decide if $r < p$. In the special case that $p = 2^{32} - 2^{30} + 1$ used in our implementation, we provide a more efficient approach to determine if $r < p$. Particularly, the parties do the following:

1) During executing sub-protocol Π_{B2A}, all parties store the arithmetic sharings $\langle \boldsymbol{r}[j] \rangle^a$ for $j \in [0, \ell - 1]$.
2) The parties set $\langle a \rangle^a := \langle \boldsymbol{r}[\ell - 1] \rangle^a$ and $\langle b \rangle^a := \langle \boldsymbol{r}[\ell - 2] \rangle^a$, and compute $\langle c \rangle^a := \sum_{j=0}^{\ell-3} \langle \boldsymbol{r}[j] \rangle^a$.
3) The parties call the (Mult) command of $\mathcal{F}_{\mathsf{LUT}}$ on input $(\langle a \rangle^a, \langle b \rangle^a, \langle c \rangle^a)$ to obtain $\langle d \rangle^a$ with $d = a \cdot b \cdot c \in \mathbb{Z}_p$.
4) The parties run $d \leftarrow \mathsf{Open}(\langle d \rangle^a)$, and output the bit indicating $r < p$ if $d = 0$ or $r \geq p$ if $d \neq 0$.

Theorem 5. *Protocol Π_{A2B} (shown in Figure 10) securely realizes the* $\mathsf{A2B}$ *command of functionality $\mathcal{F}_{\mathsf{Conv}}$ against semi-honest adversaries in the $\mathcal{F}_{\mathsf{LUT}}$-hybrid model.*

The proof of Theorem 5 is deferred to Appendix B.4.

6. Scalable Multi-Party Garbling

We present how to generate multi-party garbled circuits (MPGCs) using a key-homomorphic additive homomorphic encryption (AHE) scheme in the private-key setting and private table lookup. Building upon this, we describe a scalable MPC protocol with linear communication complexity and $O(n)$ rounds. In Appendix A.2, we show that the BGV-AHE scheme [32] in the private-key setting is key-homomorphic.

6.1. Private-Key AHE with Key Homomorphism

We provide the definition of key-homomorphic AHE schemes in the private-key setting. Let $\mathcal{K}$ and $\mathcal{M}$ denote the secret-key space and message space, respectively. We always assume that $\mathcal{K} \subseteq \mathcal{M} \subseteq \mathbb{F}^N$ where $\mathbb{F}$ is some finite field (e.g., $\mathbb{F} = \mathbb{Z}_p$) and N is a parameter determining the length of vectors. The private-key AHE scheme with the key-homomorphic property involves the following algorithms:

- **Setup:** $pp \leftarrow \mathsf{Setup}(1^\kappa)$. The setup algorithm is defined as in the threshold HE scheme shown in Section 3.2.
- **Key Generation:** $\mathsf{sk} \leftarrow \mathsf{SecKeyGen}(pp)$. On input pp, the key-generation algorithm outputs a secret key $\mathsf{sk} \in \mathcal{K}$.
- **Encryption:** $[\![\boldsymbol{m}]\!]_{\mathsf{sk},t} \leftarrow \mathsf{Enc}_{\mathsf{sk}}(t, \boldsymbol{m})$. On input sk, a label t and a vector of messages $\boldsymbol{m} \in \mathcal{M}$, the encryption algorithm outputs a ciphertext $[\![\boldsymbol{m}]\!]_{\mathsf{sk},t}$. Suppose that the scheme is instantiated by a lattice-based AHE such as private-key BGV [32] shown in Appendix A.2. In this case, t is used to retrieve/derive a vector $\boldsymbol{a}$ and then $\boldsymbol{a}$ is used to encrypt $\boldsymbol{m}$, which has been used in [15]. When the context is clear, we simply write $[\![\boldsymbol{m}]\!]_{\mathsf{sk},t}$ as $[\![\boldsymbol{m}]\!]$.
- **Decryption:** $\boldsymbol{m} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(t, [\![\boldsymbol{m}]\!])$. On input the secret key sk, a label t and a ciphertext $[\![\boldsymbol{m}]\!]$, the decryption algorithm outputs a vector of messages $\boldsymbol{m}$.
- **Key-homomorphic [26]:** Given two ciphertexts $[\![\boldsymbol{m}_1]\!]_{\mathsf{sk}_1,t}$ and $[\![\boldsymbol{m}_2]\!]_{\mathsf{sk}_2,t}$ under different secret keys $\mathsf{sk}_1, \mathsf{sk}_2$ and the same label t, any party can locally compute a ciphertext

Protocol Π_{MPGC}

Inputs: Parties $P_1, \ldots, P_n$ have 1) a Boolean circuit $\mathcal{C}$ with the set of circuit-input wires $\mathcal{I}$, the set of circuit-output wires $\mathcal{O}$ and the set of all AND and XOR gates $\mathcal{G}$; 2) a Boolean sharing $\langle x_w \rangle^b$ for each input bit $x_w \in \{0,1\}$; 3) a key-homomorphic private-key AHE scheme equipped with $(\mathsf{SecKeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Suppose that the set of public parameters pp has been established.

Preprocessing phase for generating multi-party garbled circuit:

1) All parties call functionality $\mathcal{F}_{\mathsf{LUT}}$ to sample a random Boolean sharing $\langle \lambda_w \rangle^b$ for every wire w. Then for each wire w, every party P_i samples two secret keys $\boldsymbol{k}^i_{w,0}, \boldsymbol{k}^i_{w,1}$ by running $\mathsf{SecKeyGen}(pp)$, such that $\boldsymbol{k}^i_{w,0}[0] = 0$, and $\boldsymbol{k}^i_{w,1}[0] = 0$ if $i \neq 1$ or $\boldsymbol{k}^i_{w,1}[0] = 1$ if $i = 1$. These keys constitute two vectors of arithmetic sharings $\langle \boldsymbol{k}_{w,0} \rangle^a$ and $\langle \boldsymbol{k}_{w,1} \rangle^a$ where $\boldsymbol{k}_{w,0} = \sum_{i\in[1,n]} \boldsymbol{k}^i_{w,0}$, $\boldsymbol{k}_{w,1} = \sum_{i\in[1,n]} \boldsymbol{k}^i_{w,1}$, $\boldsymbol{k}_{w,0}[0] = 0$ and $\boldsymbol{k}_{w,1}[0] = 1$.
2) For every gate $g \in \mathcal{G}$ with input wires u, v and output wire w, the parties generate a Boolean sharing $\langle e_{w,\alpha,\beta} \rangle^b$ for all $\alpha, \beta \in \{0,1\}$, where $e_{w,\alpha,\beta} \overset{\text{def}}{=} g((\lambda_u \oplus \alpha), (\lambda_v \oplus \beta)) \oplus \lambda_w \in \{0,1\}$.
   a) If g is an AND gate, then $e_{w,\alpha,\beta} = \lambda_u\lambda_v \oplus \beta\lambda_u \oplus \alpha\lambda_v \oplus \alpha\beta \oplus \lambda_w$. In this case, all parties call the (Mult) command of functionality $\mathcal{F}_{\mathsf{LUT}}$ on input $(\langle \lambda_u \rangle^b, \langle \lambda_v \rangle^b)$ to compute a Boolean sharing $\langle \lambda_u\lambda_v \rangle^b$. Then, the parties locally compute $\langle e_{w,\alpha,\beta} \rangle^b = \langle \lambda_u\lambda_v \rangle^b \oplus \beta\langle \lambda_u \rangle^b \oplus \alpha\langle \lambda_v \rangle^b \oplus \langle \lambda_w \rangle^b \oplus \alpha\beta$ for each $\alpha, \beta \in \{0,1\}$.
   b) If g is a XOR gate, then $e_{w,\alpha,\beta} = \lambda_u \oplus \lambda_v \oplus (\alpha \oplus \beta) \oplus \lambda_w$. The parties locally compute $\langle e_{w,\alpha,\beta} \rangle^b = \langle \lambda_u \rangle^b \oplus \langle \lambda_v \rangle^b \oplus \langle \lambda_w \rangle^b \oplus (\alpha \oplus \beta)$.
3) For the output wire w of each gate $g \in \mathcal{G}$, all parties generate $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}} \rangle^a$ for all $\alpha, \beta \in \{0,1\}$ as follows:
   a) The parties define a vector of arithmetic sharings $\langle T \rangle^a$ such that $\langle T[j] \rangle^a = \langle \boldsymbol{k}_{w,j} \rangle^a$ for $j \in \{0,1\}$.
   b) If g is an AND gate, then for each $\alpha, \beta \in \{0,1\}$, all parties call the $(\mathsf{PriTabLookup})$ command of functionality $\mathcal{F}_{\mathsf{LUT}}$ on input $(\langle T \rangle^a, \langle e_{w,\alpha,\beta} \rangle^b)$ to obtain $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}} \rangle^a$.
   c) If g is a XOR gate, the parties perform the following steps:
      i) Call the $(\mathsf{MaskedPriTab})$ command of functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$ on input $\langle T \rangle^a$ to generate $\langle T' \rangle^a$ and $\langle r \rangle^b$ such that $r \in \{0,1\}$ is a random bit and $T'[j] = T[r \oplus j]$ for $j \in \{0,1\}$.
      ii) Locally compute $\langle u \rangle^b := \langle e_{w,0,0} \rangle^b \oplus \langle r \rangle^b$, and then execute $\mathsf{Open}(\langle u \rangle^b)$ to obtain $u = e_{w,0,0} \oplus r \in \{0,1\}$.
      iii) Set $\langle \boldsymbol{k}_{w,e_{w,0,0}} \rangle^a = \langle \boldsymbol{k}_{w,e_{w,1,1}} \rangle^a = \langle T' \rangle^a[u]$ and $\langle \boldsymbol{k}_{w,e_{w,0,1}} \rangle^a = \langle \boldsymbol{k}_{w,e_{w,1,0}} \rangle^a = \langle T' \rangle^a[u \oplus 1]$.
4) For the output wire w of each gate $g \in \mathcal{G}$, for each $\alpha, \beta \in \{0,1\}$, all parties compute a garbled row $gg_{w,\alpha,\beta}$ as follows:
   a) Every party P_i sets a secret key $\mathsf{sk}_i := \boldsymbol{k}^i_{u,\alpha} + \boldsymbol{k}^i_{v,\beta}$, and runs $[\![\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}} \rangle^a_i]\!] \leftarrow \mathsf{Enc}_{\mathsf{sk}_i}((g,\alpha,\beta), \langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}} \rangle^a_i)$.
   b) For each $i \neq 1$, P_i sends $[\![\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}} \rangle^a_i]\!]$ to P_1, who computes $[\![\boldsymbol{k}_{w,e_{w,\alpha,\beta}}]\!] := \sum_{i\in[1,n]} [\![\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}} \rangle^a_i]\!]$, where $[\![\boldsymbol{k}_{w,e_{w,\alpha,\beta}}]\!] = \sum_{i\in[1,n]} \mathsf{Enc}_{\mathsf{sk}_i}((g,\alpha,\beta), \langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}} \rangle^a_i) = \mathsf{Enc}_{\mathsf{sk}}((g,\alpha,\beta), \boldsymbol{k}_{w,e_{w,\alpha,\beta}})$, where $\mathsf{sk} = \sum_{i\in[1,n]} \mathsf{sk}_i$.
5) Now, P_1 obtains a garbled circuit $\mathcal{GC} = \{(gg_{w,0,0}, gg_{w,0,1}, gg_{w,1,0}, gg_{w,1,1})\}_{w\in\mathcal{W}}$, where for each $\alpha, \beta \in \{0,1\}$, $gg_{w,\alpha,\beta} = [\![\boldsymbol{k}_{w,e_{w,\alpha,\beta}}]\!]$ and $\mathcal{W}$ is the set of output wires of all gates in $\mathcal{G}$.

Online phase for evaluating multi-party garbled circuit: When $\langle x_w \rangle^b$ for all $w \in \mathcal{I}$ are known, the parties execute the following:

6) For each $w \in \mathcal{I}$, all parties locally compute $\langle e_w \rangle^b = \langle x_w \rangle^b \oplus \langle \lambda_w \rangle^b$, and then the parties run $\mathsf{Open}(\langle e_w \rangle^b)$ to obtain a masked bit $e_w = x_w \oplus \lambda_w$.
7) For each $w \in \mathcal{I}$, every party P_i with $i \neq 1$ sends $\boldsymbol{k}^i_{w,e_w}$ to P_1, who computes $\boldsymbol{k}_{w,e_w} := \sum_{i\in[1,n]} \boldsymbol{k}^i_{w,e_w}$.
8) In a topological order, for each gate $g \in \mathcal{G}$ with input wires u, v and output wire w, P_1 holds $(\boldsymbol{k}_{u,e_u}, e_u)$ and $(\boldsymbol{k}_{v,e_v}, e_v)$, and then computes $\boldsymbol{k}_{w,e_w} := \mathsf{Dec}_{\boldsymbol{k}_{u,e_u}+\boldsymbol{k}_{v,e_v}}((g, e_u, e_v), gg_{w,e_u,e_v})$ and sets $e_w := \boldsymbol{k}_{w,e_w}[0]$.
9) For each $w \in \mathcal{O}$, P_1 sets $\langle y_w \rangle^b_1 := e_w \oplus \langle \lambda_w \rangle^b_1$ and P_i for each $i \neq 1$ sets $\langle y_w \rangle^b_i := \langle \lambda_w \rangle^b_i$, and the parties output $\langle y_w \rangle^b$.

Figure 11: Protocol for generating and evaluating multi-party garbled circuits in the $(\mathcal{F}_{\mathsf{Prep-LUT}}, \mathcal{F}_{\mathsf{LUT}})$-hybrid model.

$[\![\boldsymbol{m}_3]\!]_{\mathsf{sk}_3,t} = [\![\boldsymbol{m}_1]\!]_{\mathsf{sk}_1,t} + [\![\boldsymbol{m}_2]\!]_{\mathsf{sk}_2,t}$ such that $\mathsf{sk}_3 = \mathsf{sk}_1 + \mathsf{sk}_2$ and $\boldsymbol{m}_3 = \boldsymbol{m}_1 + \boldsymbol{m}_2$.

We always assume that the AHE scheme satisfies the standard correctness. We need that the AHE scheme satisfies a simple variant of the CPA security. Specifically, any PPT adversary $\mathcal{A}$ can make a query $(\boldsymbol{m}_i, t_i)$ to the encryption oracle which returns $[\![\boldsymbol{m}_i]\!]_{\mathsf{sk},t_i}$ to $\mathcal{A}$ for each $i \in [1, \ell]$ where ℓ is the number of oracle queries, and then $\mathcal{A}$ can choose two message-label pairs $(\boldsymbol{m}^*_0, t^*_0)$ and $(\boldsymbol{m}^*_1, t^*_1)$ with $t^*_0, t^*_1 \notin \{t_1, \ldots, t_\ell\}$. Then the probability that $\mathcal{A}$ distinguishes $[\![\boldsymbol{m}^*_0]\!]_{\mathsf{sk},t^*_0}$ from $[\![\boldsymbol{m}^*_1]\!]_{\mathsf{sk},t^*_1}$ is negligible in κ. Furthermore, the CPA security holds for a polynomial number of secret keys, which is guaranteed using a standard hybrid argument. When a lattice-based AHE scheme is adopted, t uniquely determines $\boldsymbol{a}$ used in encryption, and the security notion is naturally equivalent to the standard CPA security.

In the private-key setting, we show that the BGV scheme with a single level [32] is a key-homomorphic AHE scheme, which is described in Appendix A.2. Alternatively, the BFV-AHE scheme [33], [34] is another candidate.

6.2. Multi-Party Garbled Circuits

In Figure 11, we give the details of the MPC protocol based on multi-party garbled circuits. Without loss of generality, we assume that only one party P_1 can evaluate the garbled circuit, which is easy to be extended to support that all parties are able to evaluate the garbled circuit. To be compatible with the conversion protocols between arithmetic sharings and Boolean sharings shown in Section 5, we consider that the inputs and outputs of the parties are Boolean sharings. In a general case that the inputs are secret bits, the parties can run the Share algorithm to generate corresponding Boolean sharings. To make P_1 get the output, the parties can send the shares of the Boolean sharings on circuit-output wires to P_1 who reconstructs the output bits by running the Rec algorithm. We can use a standard approach to support that all parties obtain different outputs.

We divide the MPC protocol into two phases: the preprocessing phase and online phase. In the preprocessing phase, for each wire w, the parties generate the Boolean sharing of a random mask $\langle\lambda_w\rangle^b$, and we refer to $e_w = z_w \oplus \lambda_w$ as a masked value where $z_w \in \{0,1\}$ is an actual value for w. For each wire w, every party P_i samples two random keys $\boldsymbol{k}^i_{w,0}, \boldsymbol{k}^i_{w,1}$ for the private-key AHE scheme, where the sums of these keys are the keys $\boldsymbol{k}_{w,0}, \boldsymbol{k}_{w,1}$ encrypted in the garbled circuit. As in prior work [15], we set the first components of vectors $\boldsymbol{k}_{w,0}, \boldsymbol{k}_{w,1}$ are 0 and 1 respectively, which allows the evaluator to extract the masked value e_w from $\boldsymbol{k}_{w,e_w}$. At first glance, this loses one dimension of the secret key, which slightly reduces security. On the one hand, when using BGV as the AHE scheme, the secret keys can be sampled uniformly from R_3, and have already sufficiently high entropy to guarantee the security. On the other hand, as we need only private-key AHE, the secret keys can actually be sampled uniformly from R_q (instead of R_3) where $q \gg 3$. For each gate with output wire w, the parties also compute the Boolean sharings of four masked values $\langle e_{w,\alpha,\beta}\rangle^b$ for all $\alpha, \beta \in \{0,1\}$, where α, β enumerate all four possible values of masked values on input wires of the gate. These Boolean sharings $\langle e_{w,\alpha,\beta}\rangle^b$ for each $\alpha, \beta \in \{0,1\}$ can be used to generate arithmetic sharings $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$ using our table-lookup approach. Through the key-homomorphic addition operations of the private-key AHE scheme, for each gate with input wires u, v and output wire w, the parties jointly compute the garbled rows like the classical Yao's GC.

In the online phase, for each circuit-input wire w, all parties open e_w, and then every party sends $\boldsymbol{k}^i_{w,e_w}$ to P_1 who reconstructs the key $\boldsymbol{k}_{w,e_w}$. Then, P_1 can evaluate the garbled circuit by decrypting the corresponding garbled rows. Finally, for each circuit-output wire w, the parties can locally compute a Boolean sharing $\langle y_w\rangle^b$. In the following theorem, we prove that protocol Π_{MPGC} securely realizes the standard MPC functionality $\mathcal{F}_{\mathsf{MPC}}$.

Theorem 6. *Protocol* Π_{MPGC} *(shown in Figure 11) securely realizes functionality* $\mathcal{F}_{\mathsf{MPC}}$ *against semi-honest adversaries in the* $(\mathcal{F}_{\mathsf{Prep-LUT}}, \mathcal{F}_{\mathsf{LUT}})$*-hybrid model, assuming that the key-homomorphic AHE scheme is CPA secure.*

The proof of Theorem 6 can be found in Appendix B.5.

7. Implementation Optimizations

We discuss two optimizations for the preprocessing protocols of masked lookup tables $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ (Figure 6) and $\Pi^{\mathsf{polysize}}_{\mathsf{prepLUT}}$ (Figure 7), along with one optimization for the scalable MPC protocol Π_{MPGC} (Figure 11).

7.1. Optimizations for Lookup Table Protocol

Bootstrapping. In protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ (shown in Figure 6), we use the packing technique to reduce the communication cost incurred by ciphertext expansion, which in turn requires a customized private pair-wise swapping. To obtain the communication complexity linear in the number of parties, the underlying THE scheme requires a budget to support the number of scale-multiplication and rotation operations that is linear in the number of all parties. However, for a large number of parties, it would require a very large size of parameters. We use bootstrapping to refrain from having a large budget. We consider two approaches to realize bootstrapping for threshold HE.

1) *Non-interactive bootstrapping:* All parties run an MPC protocol to generate a bootstrapping key, and then use the key to bootstrap HE ciphertexts and thus reduce the noise size.
2) *Interactive bootstrapping:* The parties can convert an evaluated HE ciphertext to a vector of arithmetic sharings, and then convert it back to a fresh HE ciphertext, following the previous work [29].

The non-interactive bootstrapping allows us to achieve communication complexity linear in the number of parties. Our implementation adopts interactive bootstrapping as it is more efficient for most of the reasonable network configurations. However, the communication complexity is now quadratic in the number of parties with a very small constant (≈ 0.1). Both bootstrapping approaches as described above can also be applied in protocol $\Pi^{\mathsf{polysize}}_{\mathsf{prepLUT}}$ (Figure 7) in the same way.

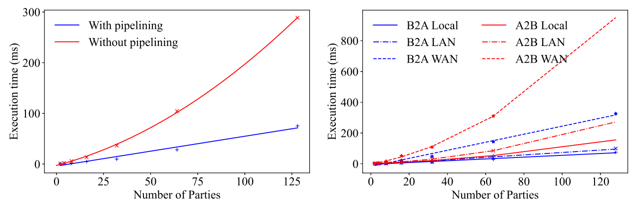
Pipelining. Our protocols (i.e., $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ and $\Pi^{\mathsf{polysize}}_{\mathsf{prepLUT}}$) do not require a party to remain active after it sends a HE ciphetext to the next party. This enables us to pipeline the computation, such that the computation complexity is linear in the number of parties rather than quadratic. Instead of processing all the ciphertexts at once and sitting idle, each party processes one ciphertext, then sends it to the next party and starts processing the next ciphertext. For a small number of parties where bootstrapping is not required, this method of pipelining works.

When interactive bootstrapping is involved, the parties are idle but waiting for a bootstrapping request to respond. To handle such case, we use the system called `poll`. On one thread, every party executes the protocol regularly as if the ring structure works (i.e., receiving a ciphertext from the previous party, processing it, and sending it to the next party), and on the other thread, it listens for any ready bootstrapping request and responds to it. We allow the maximum multiplicative depth for the underlying THE scheme to be 10, and choose the minimum budget that requires the same number of bootstrapping requests as with a budget of 10.

We evaluate the performance of the protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ with and without pipelining. The performance evaluation is reported in Figure 12a. As shown in the figure, with pipelining the running time scales linearly with the number of parties rather than quadratically as without pipelining.

7.2. Optimization for Multi-Party Garbling

In the MPGC protocol (shown in Figure 11), every garbler P_i with $i \neq 1$ sends four AHE ciphertexts for each gate to the evaluator P_1, who combines all the ciphertexts



(a) $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ (b) Π_{B2A} and Π_{A2B}

Figure 12: **Microbenchmarks of our protocols.** (a) The running time of protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ with and without pipelining optimization. (b) Running time of the offline phase of our Boolean-to-arithmetic and arithmetic-to-Boolean conversion protocols.

into a garbled circuit. When the circuit is large or the number of parties n is large, P_1 needs a very large bandwidth to receive these ciphertexts. When the bandwidth of P_1 is not sufficient, this would form an efficiency bottleneck. To solve the efficiency issue, we adopt a different communication pattern. That is, we let P_n send the AHE ciphertexts to P_{n-1}, who combines these ciphertexts with the AHE ciphertexts computed by itself; and then P_{n-1} sends the resulting ciphertexts to P_{n-2} and so on. This amortizes the $O(n)$ communication of P_1 to constant communication for every party. Nevertheless, this increases the round complexity from one round to $n-1$ rounds. To reduce the rounds, we can adopt a binary-tree architecture to transfer AHE ciphertexts. In particular, all the parties are arranged in a binary tree such that each node only interacts with its children and parent nodes. Two children nodes P_i and P_{i+1} send the AHE ciphertexts to their parent node P_{i+2}, who aggregates the AHE ciphertexts from three parties and then sends the resulting ciphertexts to the parent node of P_{i+2}. The communication bandwidth of every party is at most $2\times$ larger than the first approach, and the rounds are reduced from $n-1$ to $\log n$. In addition, the AHE ciphertexts generated by every party P_i can be sent in a pipelined way.

8. Performance Evaluation

8.1. Summary of Evaluation

We summarize the key findings from our performance evaluation below.

1) We show that pipelining protocol $\Pi^{\mathsf{size2}}_{\mathsf{prepLUT}}$ (Figure 6) improves its execution time from quadratic in the number of parties to almost linear in the number of parties.
2) We compare our conversion protocols with the state-of-the-art works MOTION [11] and MP-SPDZ [21].
   a) In terms of running time, for 64 parties, our protocols improve MP-SPDZ by a factor of $2247\times$ for B2A conversion and $369\times$ for A2B conversion.
   b) In terms of communication cost, for 64 parties, our protocols improve MP-SPDZ by a factor of $15384\times$ for B2A conversion and $8819\times$ for A2B conversions; for 32 parties, we improve MOTION by factor of $20\times$ for B2A and $1184\times$ for A2B.
3) When applying our protocol on end-to-end applications, we achieve $8242\times$ improvements in communication and up to $1490\times$ improvements in monetary cost.
4) Protocol Π_{MPGC} (Figure 11) reduces the inbound communication per party of optimized BMR [20] by about 56 GB for evaluating an AES circuit among 128 parties. Note that the inbound communication is an efficiency bottleneck of multi-party distributed garbling, as a central party receives garbled circuits from all other parties.

8.2. Evaluation Setup

We implemented our protocols using EMP-toolkit [43] for correlated OT and OpenFHE [44] for threshold homomorphic encryption. The implementation will be open-sourced, and we are happy to provide it upon request. All experiments are conducted on AWS of instance type `m5.2xlarge`. We consider three settings with up to 128 parties, which are described as follows:

1) **Local setting:** The network bandwidth is up to 10 Gbps with 0.1 ms latency.
2) **LAN setting:** The network bandwidth is up to 1 Gbps with 0.1 ms latency.
3) **WAN setting:** The network bandwidth is up to 200 Mbps with 100 ms latency.

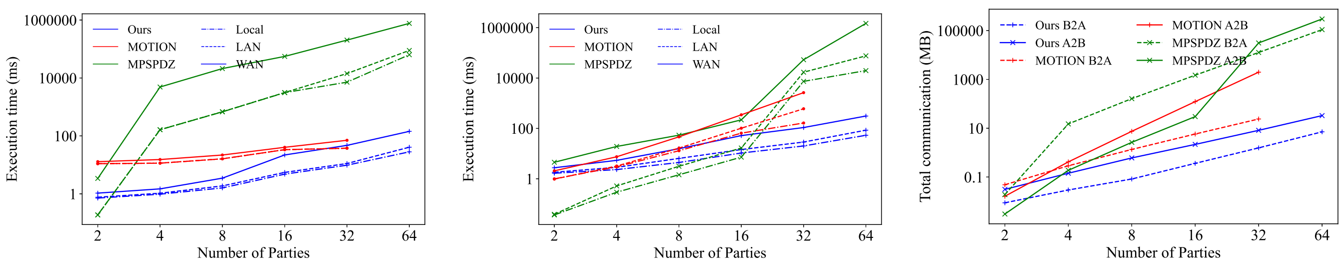
For threshold HE in the public-key setting, we choose the parameters that achieve the 128-bit security level [45]. The plaintext prime p is equal to $2^{32}-2^{30}+1$, the length of the ciphertext prime q is more than 530 bits, and the number of slots $N = 65536$.

8.3. Performance of Conversions

We evaluate the performance of conversion protocols Π_{A2B} and Π_{B2A} in the offline phase and the online phase, respectively. We compare the performance of our conversion protocols with the state-of-the-art protocols MOTION [11] and MP-SPDZ [21] for semi-honest security with all-but-one corruption. When using `m5.2xlarge`, MOTION can only execute up to 16 parties due to their high requirement of hardware resources. When we increase the instance size to `m5.4xlarge`, MOTION can be successfully executed with 32 parties; to further scale MOTION with 64 parties, one would need to use even larger machines. We note that our framework relies on the RLWE assumption, while MOTION (and MP-SPDZ, resp.) depend on LPN (and Minicrypt, resp.).

Execution time of offline phase. We evaluate the performance of the offline phase for our conversion protocols up to 128 parties in all three settings. The performance for Π_{B2A} and Π_{A2B} is reported in Figure 12b, which shows that the performance is linear in the number of parties. The execution time for the WAN setting grows faster than that in the LAN setting due to the communication overhead.

We compare the execution time with MOTION and MP-SPDZ for B2A and A2B conversions in Figure 13a and Figure 13b, respectively. We observe that our framework outperforms the existing conversion protocols in all three settings. Compared to MP-SPDZ, our B2A (resp., A2B) protocol improves the running time of the offline phase by a factor of $2247\times$ (resp., $369\times$) for 64 parties. Compared to



(a) Offline phase of Π_{B2A}. (b) Offline phase of Π_{A2B}. (c) Total communication.

Figure 13: **Performance comparison for the conversion protocols of our framework, MOTION [11] and MP-SPDZ [21].** For the MOTION benchmarks for 32 parties, each party runs on a machine with double the resources than all other benchmarks.

Protocol	Setting	2	4	8	16	32	64
B2A	Local	0.038	0.046	0.05	0.066	0.11	0.494
	LAN	0.038	0.049	0.06	0.09	0.157	3.461
	WAN	0.066	0.076	0.181	0.357	0.988	4.696
A2B	Local	0.012	0.002	0.003	0.004	0.006	0.05
	LAN	0.006	0.004	0.006	0.012	0.023	0.049
	WAN	0.157	0.087	0.12	0.147	0.209	0.333

TABLE 1: **Performance of the online phase of our conversion protocols.** Running time is measured in milliseconds (ms), and the first row (2∼64) is the number of parties. Running time is amortized over many conversions.

Triples	Setting	2	4	8	16	32	64
Boolean	Local	0.7	3.6	8.2	8.1	17	34.2
	LAN	0.7	3.5	8.5	9.3	18.6	37.2
	WAN	0.7	3.5	14.9	24.1	49.2	99.5
Arithmetic	Local	11.4	26.7	46.7	61	94.6	174.9
	LAN	12.6	27.9	63.4	109.3	191.1	418.2
	WAN	11.4	27.9	128	312.2	674	1621.5

TABLE 2: **Performance of Beaver triple generation.** Running time in microseconds (μs) for Boolean sharings (using Ferret-COT) and arithmetic sharings (using BGV THE).

MOTION, in the offline phase, the running-time improvement of our B2A (resp., A2B) protocol is up to $15\times$ (resp., $20\times$) for 32 parties.

Execution time of online phase. The online phase of our conversion protocols is interchangeable with that of MOTION and MP-SPDZ. We evaluate the online running time for our protocols in all three network settings. Benchmarks for the online time can be found in Table 1. We note that the running time of the online phase is at most 10% of the offline time.

Communication cost of conversions. We benchmark the total communication required by each conversion using our protocols, MOTION and MP-SPDZ. The comparison of the total communication cost is reported in Figure 13c. The multi-party LUT protocol used as the main building block in our conversion protocols does not require parties to communicate with every other party; thus, our system requires a significantly smaller amount of communication — up to $20\times$ less than MOTION and $15384\times$ less than MP-SPDZ for B2A and $1184\times$ less than MOTION and $8819\times$ less than MP-SPDZ for A2B.

8.4. Performance of Boolean/Arithmetic Triples

For completeness, we benchmark the performance of triple generation for arithmetic and Boolean circuits in Table 2. With increasing number of parties, we observe that the cost of arithmetic triple generation increases slower than the cost of Boolean triple generation. This is because we use COT to generate the Boolean triples. This can be replaced by THE. However, using THE increases the computational overhead significantly.

#Parties	Biomatch	Kmeans	MNIST	Gauss Dist.	Merge DB
4	1.93	6.16	25.49	0.07	0.24
16	9.7	17.79	94.44	0.27	1.18
64	64.97	76.6	387.7	1.52	8.51

TABLE 3: **Performance of end-to-end applications.** Performance estimation in seconds (s) for running the end-to-end applications in the LAN setting.

8.5. Performance for End-to-End Applications

We estimate the performance of several end-to-end applications using mixed-mode circuits generated by Silph [46]. The applications are listed with the number of operations in Table 5. The performance estimation to run the end-to-end applications is given in Table 3.

Monetary cost analysis. We analyze the monetary cost of running the applications among 64 parties using our framework and MP-SPDZ. Suppose that we run the instances in a single region, and the communication cost is USD 0.01 per GB. We run each party on AWS of instance type `m5.2xlarge`, which costs USD 0.384 per hour. A detailed comparison is given in Table 4. We observe that MP-SPDZ is up to $1490\times$ more expensive than our framework.

Protocol	Biomatch	Kmeans	MNIST	Gauss Dist.	Merge DB
Ours	0.54	1.55	2.97	0.02	0.08
MP-SPDZ	803.3	399.8	102.45	21.75	10.05

TABLE 4: **Comparison of monetary cost for our framework and MP-SPDZ [21].** The monetary cost estimates in USD for running the applications among 64 parties.

Applications	#AND	#MULT	#B2A	#A2B
Biomatch	23,205	1024	1028	256
Kmeans	1,568,660	8800	118	116
MNIST	1,624,460	666,600	1192	1
Gauss Dist.	8275	14	15	7
Merge DB	3930	200	201	1

TABLE 5: **The number of operations using mixed-mode circuits for end-to-end applications.**

8.6. Communication Cost of MPGC

The OpenFHE library [44] does not support private-key AHE, and so we only give a conservative estimation of the communication cost of our protocol Π_{MPGC} (Figure 11). For the parameters with 128-bit security level, we select the plaintext prime p to be $2^{16}+1$, the length of the ciphertext prime q is more than 45 bits, and the number of slots $N = 4096$. We find that our protocol has constant inbound and outbound communication per party. We report the inbound and outbound communication per party when securely computing an AES-128 circuit with 128 parties using our protocol Π_{MPGC} and the optimized BMR protocol [20]. Both inbound and outbound communication for Π_{MPGC} is 51.12 GB. The inbound and outbound communication for BMR is 107.37 GB and 0.33 GB, respectively. Compared to BMR, the inbound communication per party for Π_{MPGC} is lower, while the outbound communication is higher.

Acknowledgements

Work of Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019). Work of Jonathan Katz was partially supported by NSF award #1837517. Work of Radhika and Xiao Wang is supported by NSF awards #2016240, #2318974 #2236819 and research awards from Meta, Google, and JPMorgan Chase & Co. Any views or opinions expressed herein are solely those of the authors listed, and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction.

References

[1] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: tool for automating secure two-party computations," in *ACM Conf. on Computer and Communications Security (CCS) 2010*. ACM Press, 2010. 1

[2] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *Network and Distributed System Security Symposium*. The Internet Society, 2015. 1

[3] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation," in *USENIX Security Symposium 2021*. USENIX Association, 2021. 1

[4] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," Cryptology ePrint Archive, Report 2020/721, 2020, https://eprint.iacr.org/2020/721. 1

[5] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *ACM Conf. on Computer and Communications Security (CCS) 2018*. ACM Press, 2018. 1

[6] A. Patra and A. Suresh, "BLAZE: Blazing fast privacy-preserving machine learning," in *Network and Distributed System Security Symposium*. The Internet Society, 2020. 1

[7] A. P. K. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *USENIX Security Symposium 2021*. USENIX Association, 2021. 1

[8] D. Rotaru and T. Wood, "MArBled circuits: Mixing arithmetic and Boolean circuits with active security," in *Progress in Cryptology—Indocrypt*, ser. LNCS. Springer, 2019. 1, 2.1

[9] I. Damgård, D. Escudero, T. K. Frederiksen, M. Keller, P. Scholl, and N. Volgushev, "New primitives for actively-secure MPC over rings with applications to private machine learning," in *IEEE Symp. Security and Privacy 2019*. IEEE, 2019. 1, 2.1

[10] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved primitives for MPC over mixed arithmetic-binary circuits," in *Advances in Cryptology—Crypto 2020, Part II*, ser. LNCS. Springer, 2020. 1, 2.1, 3.3

[11] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko, "Motion – a framework for mixed-protocol multi-party computation," *ACM Trans. Priv. Secur.*, mar 2022. 1, 4, 2.1, 2, 8.3, 13

[12] N. Koti, S. Patil, A. Patra, and A. Suresh, "Mpclan: Protocol suite for privacy-conscious computations," *Journal of Cryptology*, vol. 36, no. 3, p. 22, 2023. 1

[13] I. Damgård and Y. Ishai, "Constant-round multiparty computation using a black-box pseudorandom generator," in *Advances in Cryptology—Crypto 2005*, ser. LNCS, vol. 3621. Springer, 2005. 1

[14] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant round multi-party computation combining BMR and SPDZ," in *Advances in Cryptology—Crypto 2015, Part II*, ser. LNCS, vol. 9216. Springer, 2015. 1

[15] A. Ben-Efraim, Y. Lindell, and E. Omri, "Efficient scalable constant-round MPC via garbled circuits," in *Advances in Cryptology—Asiacrypt 2017, Part II*, ser. LNCS. Springer, 2017. 1, 2.3, 6.1, 6.2, A.2

[16] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *ACM Conf. on Computer and Communications Security (CCS) 2017*. ACM Press, 2017. 1, 3

[17] C. Hazay, E. Orsini, P. Scholl, and E. Soria-Vazquez, "Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT)," in *Advances in Cryptology—Asiacrypt 2018, Part III*, ser. LNCS. Springer, 2018. 1

[18] A. Ben-Efraim, K. Cong, E. Omri, E. Orsini, N. P. Smart, and E. Soria-Vazquez, "Large scale, actively secure computation from LPN and free-XOR garbled circuits," in *Advances in Cryptology—Eurocrypt 2021, Part III*, ser. LNCS. Springer, 2021. 1

[19] G. Beck, A. Goel, A. Hegde, A. Jain, Z. Jin, and G. Kaptchuk, "Scalable multiparty garbling," Cryptology ePrint Archive, Report 2023/099, 2023, https://eprint.iacr.org/2023/099. 1

[20] A. Ben-Efraim, Y. Lindell, and E. Omri, "Optimizing semi-honest secure multiparty computation for the internet," in *ACM Conf. on Computer and Communications Security (CCS) 2016*. ACM Press, 2016. 3, 4, 8.6

[21] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *ACM Conf. on Computer and Communications Security (CCS) 2020*. ACM Press, 2020. 4, 2, 8.3, 13, 4

[22] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round OT extension and silent non-interactive secure computation," in *ACM Conf. on Computer and Communications Security (CCS) 2019*. ACM Press, 2019. 2.1, 3.3

[23] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *ACM Conf. on Computer and Communications Security (CCS) 2020*. ACM Press, 2020. 2.1, 3.3

[24] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE," in *Advances in Cryptology—Asiacrypt 2017, Part I*, ser. LNCS. Springer, 2017. 2.1

[25] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *41st Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2009. 2.2, A.1

[26] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *Advances in Cryptology—Eurocrypt 2012*, ser. LNCS. Springer, 2012. 2.2, 3.2, 6.1, A.2

[27] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptology*, vol. 13, no. 1, Jan. 2000. 3

[28] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Advances in Cryptology—Crypto 2018, Part I*, ser. LNCS, vol. 10991. Springer, 2018. 3.2

[29] C. Mouchet, J. R. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *PoPETs*, vol. 2021, no. 4, Oct. 2021. 3.2, 2, A.1, A.3

[30] F. Bourse, R. del Pino, M. Minelli, and H. Wee, "FHE circuit privacy almost for free," in *Advances in Cryptology—Crypto 2016, Part II*, ser. LNCS, vol. 9815. Springer, 2016. 3.2

[31] L. de Castro, C. Juvekar, and V. Vaikuntanathan, "Fast vector oblivious linear evaluation from ring learning with errors," Cryptology ePrint Archive, Report 2020/685, 2020, https://eprint.iacr.org/2020/685. 3.2

[32] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*. Cambridge, MA, USA: Association for Computing Machinery, Jan. 8–10, 2012. 3.2, 6, 6.1, A.1, A.2

[33] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology—Crypto 2012*, ser. LNCS, vol. 7417. Springer, 2012. 3.2, 6.1

[34] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012, https://eprint.iacr.org/2012/144. 3.2, 6.1

[35] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology—Eurocrypt 2010*, ser. LNCS. Springer, 2010. 3.2

[36] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *ACM Conf. on Computer and Communications Security (CCS) 2016*. ACM Press, 2016. 3.3

[37] M. Keller, E. Orsini, D. Rotaru, P. Scholl, E. Soria-Vazquez, and S. Vivek, "Faster secure multi-party computation of AES and DES using lookup tables," in *Intl. Conference on Applied Cryptography and Network Security (ACNS)*, ser. LNCS. Springer, 2017. 4, 4.3

[38] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky, "On the power of correlated randomness in secure computation," in *9th Theory of Cryptography Conference—TCC 2013*, ser. LNCS, vol. 7785. Springer, 2013. 4.3

[39] I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci, "The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited," in *Advances in Cryptology—Crypto 2017, Part I*, ser. LNCS, vol. 10401. Springer, 2017. 4.3

[40] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *Network and Distributed System Security Symposium*. The Internet Society, 2017. 4.3

[41] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, and H. Yalame, "Flute: Fast and secure lookup table evaluations," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 515–533. 4.3

[42] R. Garg, K. Yang, J. Katz, and X. Wang, "Scalable mixed-mode mpc," Cryptology ePrint Archive, Paper 2023/1700, 2023, https://eprint.iacr.org/2023/1700. 5.1

[43] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016. 8.2

[44] A. A. Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-source fully homomorphic encryption library," Cryptology ePrint Archive, Report 2022/915, 2022, https://eprint.iacr.org/2022/915. 8.2, 8.6

[45] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption standard," Cryptology ePrint Archive, Report 2019/939, 2019, https://eprint.iacr.org/2019/939. 8.2

[46] E. Chen, J. Zhu, A. Ozdemir, R. S. Wahby, F. Brown, and W. Zheng, "Silph: A framework for scalable and accurate generation of hybrid MPC protocols," 2023. 8.5

[47] S. Halevi and V. Shoup, "Bootstrapping for HElib," in *Advances in Cryptology—Eurocrypt 2015, Part I*, ser. LNCS, vol. 9056. Springer, 2015. A.1

Appendix A. BGV Homomorphic Encryption

A.1. Public-Key BGV Full-Threshold HE Scheme

We outline a full-threshold version of the public-key BGV HE scheme. We refer the reader to [32], [47], [29] for more details (e.g., rotation and bootstrapping). The set of public parameters pp defines the following parameters:

- The number of slots N. The plaintext modulus p and the ciphertext modulus q such that p and q are co-prime and $n \cdot p < q$ where n is the number of parties, i.e., the plaintext and ciphertext are defined in R_p and R_q respectively.
- The standard variance σ defines a discrete Gaussian distribution $\chi(\sigma)$. For circuit privacy based on noise flooding, σ_{cp} defines another discrete Gaussian distribution $\chi(\sigma_{\mathsf{cp}})$, where σ_{cp} is exponentially larger than σ.
- The polynomial ring R_3 where the coefficients are picked from $\{-1, 0, 1\}$. Let $\mathcal{Z}$ be the distribution in which sampling one polynomial in R_3 such that each coefficient is 1 with probability $1/4$, -1 with probability $1/4$ and 0 with probability $1/2$.
- Let $\mathcal{H}$ be the distribution sampling one polynomial in R_3 such that at least h coefficients are non-zero for some parameter h. A common random polynomial $a \leftarrow R_q$.

Given the above public parameters, the BGV-THE scheme has the following algorithms:

- $\mathsf{SecKeyGen}(pp)$: Each party P_i samples $s_i \leftarrow \mathcal{H}$ and sets $\mathsf{sk}_i = s_i$.

- $\Pi_{\mathsf{PubKeyGen}}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$: Each party P_i samples $e_i \leftarrow \chi(\sigma)$ and computes $b_i := -a \cdot s_i + p \cdot e_i \in R_q$. For each $i \neq 1$, P_i sends b_i to P_1. Then, P_1 computes $b := \sum_{i=1}^{n} b_i \in R_q$ and sends b to all other parties. The parties $P_1, \ldots, P_n$ output $\mathsf{pk} = (a, b)$.
- $\mathsf{Enc}_{\mathsf{pk}}(m)$: To encrypt a message $m \in R_p \cong (\mathbb{Z}_p)^N$, sample $v \leftarrow \mathcal{Z}, e_0, e_1 \leftarrow \chi(\sigma)$, and compute $c_0 := b \cdot v + p \cdot e_0 + m \in R_q$ and $c_1 := a \cdot v + p \cdot e_1 \in R_q$. Output a ciphertext $[\![m]\!] = (c_0, c_1)$.
- $\mathsf{Dec}_{\mathsf{sk}}([\![m]\!])$: On input a secret key $\mathsf{sk} = \sum_{i \in [1,n]} \mathsf{sk}_i$ and $[\![m]\!] = (c_0, c_1)$, write $s = \mathsf{sk}$, compute $m' := c_0 + s \cdot c_1 \in R_q$ and set $m := m' \mod p$. Protocol $\Pi_{\mathsf{Dec}}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, [\![m]\!])$ is not used in our lookup table protocols, and thus is omitted.

Circuit privacy with noise flooding [25]. The transformation algorithm $\mathsf{CP}([\![m]\!])$ is performed as follows:

1) Sample two large noises $e'_0, e'_1 \leftarrow \chi(\sigma_{\mathsf{cp}})$, and then run $[\![0]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0; (e'_0, e'_1))$, where Enc adopts e'_0, e'_1 (instead of sampling noises from $\chi(\sigma)$) to encrypt zero.
2) Output a circuit-private ciphertext $ct := [\![m]\!] + [\![0]\!]$.

The above approach can transform an evaluated ciphertext to a ciphertext with circuit privacy, using exponentially large noises to flood the noises underlying the ciphertext $[\![m]\!]$.

A.2. Private-Key Key-Homomorphic BGV-AHE

Below, we outline the private-key BGV-AHE scheme with the key-homomorphic property. We refer the reader to [32] for more details. Let $\{t_1, \ldots, t_\ell\}$ be the set of all possible labels. As such, the set of public parameters pp in the private-key setting also define the parameters p, q, N and the error distribution $\chi(\sigma)$. Differently, pp now defines a set of random polynomials $a_1, \ldots, a_\ell \leftarrow R_q$, where each polynomial a_i corresponds to a label t_i. If the set of labels is large, then the size of pp is very large. We have the following two approaches to solve the issue:

- Let $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to R_q$ be a random oracle. Sample a random key $\mathsf{key} \leftarrow \{0,1\}^\kappa$, and any party can compute $a_i := \mathsf{F}(\mathsf{key}, t_i)$ for each $i \in [1, \ell]$. Now, pp only needs to involve key, but this approach adds a random-oracle computation for each encryption.
- For the application of the MPGC protocol (described in Section 6), each label t_i corresponds to a triple (g, α, β) where g is a gate and $\alpha, \beta \in \{0,1\}$. In this application, following the work [15], we can only put random polynomials $a_1, \ldots, a_{8 \cdot f_{out}} \in R_q$ into pp, where f_{out} is the maximal fan-out of the circuit. Using the algorithm in [15], one can assign a random polynomial a_i into the encryption of each garbled row, such that any two gates sharing an input wire do not share any of the random polynomials. In this way, pp includes $8 \cdot f_{out}$ polynomials in R_q but less computation is required.

In the following, we give the construction of the private-key BGV-AHE scheme.

- $\mathsf{SecKeyGen}(pp)$: Sample $s \leftarrow R_q$ and output $\mathsf{sk} = s$.
- $\mathsf{Enc}_{\mathsf{sk}}(t, m)$: On input a message $m \in R_p$, a label t and a secret key $\mathsf{sk} = s$, retrieve a from pp according to t, sample $e \leftarrow \chi(\sigma)$ and compute $c := a \cdot s + p \cdot e + m \in R_q$. Output a ciphertext $[\![m]\!] = c$.
- $\mathsf{Dec}_{\mathsf{sk}}(t, [\![m]\!])$: On input a ciphertext $[\![m]\!] = c$, a label t and $\mathsf{sk} = s$, retrieve a from pp based on t, compute $m' := c - a \cdot s \in R_q$, and output $m := m' \mod p$.
- **Key-homomorphic addition:** Given two ciphertexts $c_1 = a \cdot s_1 + p \cdot e_1 + m_1$ and $c_2 = a \cdot s_2 + p \cdot e_2 + m_2$, any party can compute $c_3 := c_1 + c_2 = a \cdot (s_1 + s_2) + p \cdot (e_1 + e_2) + (m_1 + m_2)$. Let $s_3 = s_1 + s_2$, $e_3 = e_1 + e_2$ and $m_3 = m_1 + m_2$. Then $c_3 = a \cdot s_3 + p \cdot e_3 + m_3$.

Under the ring-LWE assumption, it is easy to prove that the above private-key BGV-AHE scheme satisfies the CPA security (in Section 6.1), following prior works [32], [26].

Protocol Π_{E2A}

Inputs: Parties $P_1, \ldots, P_n$ hold the following inputs:

- The set of public parameters pp for public-key BGV-THE.
- A ciphertext $[\![\boldsymbol{x}]\!] = (c_0, c_1)$ with $\boldsymbol{x} \in R_p$ and $c_0, c_1 \in R_q$.
- P_i holds a share of the secret key $\mathsf{sk}_i = s_i$.

Conversion from encryption to arithmetic sharings:

1) For each $i \neq 1$, P_i samples $e_i \leftarrow \chi(\sigma_{\mathsf{cp}})$ and $\langle \boldsymbol{x} \rangle_i^a \leftarrow R_p$, and then sends $h_i := s_i \cdot c_1 + p \cdot e_i - \langle \boldsymbol{x} \rangle_i^a \in R_q$ to P_1.
2) The party P_1 computes $\langle \boldsymbol{x} \rangle_1^a := s_1 \cdot c_1 + c_0 + \sum_{i=2}^{n} h_i$.
3) All parties output $\langle \boldsymbol{x} \rangle^a$ with $\boldsymbol{x} \in (\mathbb{Z}_p)^N$.

Figure 14: Protocol for converting BGV HE encryption into arithmetic sharings.

A.3. Conversions between BGV-THE Encryption and Arithmetic Sharings

We show how to convert between the BGV ciphertexts and arithmetic sharings, where the public-key BGV-THE scheme (shown in Appendix A.1) is used for encryption. In Figure 14, we describe the conversion protocol from BGV public-key encryption to arithmetic sharings. This protocol follows the protocol in [29], except for replacing BFV THE with BGV THE. In Figure 14, the noise e_i is sampled from a discrete Gaussian distribution $\chi(\sigma_{\mathsf{cp}})$, which is used to keep s_i private based on noise flooding. The BGV-THE scheme supports the packing technique, and thus a single ciphertext encrypts a vector in $(\mathbb{Z}_p)^N$ and the protocol would output a vector of arithmetic sharings. It is easy to prove that protocol Π_{E2A} (Figure 14) securely realizes the (E2A) command of functionality $\mathcal{F}_{\mathsf{Conv}}$ (Figure 3) under the ring-LWE assumption following the work [29].

For conversion from arithmetic sharings to encryption of BGV-THE, the protocol is constructed as follows:

1) $P_1, \ldots, P_n$ are given a vector of arithmetic sharings $\langle \boldsymbol{x} \rangle^a$ with $\boldsymbol{x} \in (\mathbb{Z}_p)^N$ and a public key pk for BGV-THE.
2) For $i \in [1, n]$, P_i encodes $\langle \boldsymbol{x} \rangle_i^a$ into a polynomial in R_p and then runs $[\![\langle \boldsymbol{x} \rangle_i^a]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\langle \boldsymbol{x} \rangle_i^a)$.
3) For $i \neq 1$, P_i sends $[\![\langle \boldsymbol{x} \rangle_i^a]\!]$ to P_1. Then, P_1 computes $[\![\boldsymbol{x}]\!] := \sum_{i \in [1,n]} [\![\langle \boldsymbol{x} \rangle_i^a]\!]$ and sends it to all other parties.

It is easy to see that if the BGV-THE scheme is CPA secure, then the above protocol securely realizes the (A2E) command of functionality $\mathcal{F}_{\mathsf{Conv}}$ (shown in Figure 3).

Appendix B. Formal Proofs of Our Protocols

For all our proofs, we use $\mathcal{H}$ and $\mathcal{M}$ to denote the set of honest parties and the set of corrupted parties, respectively.

B.1. Proof of Theorem 1

Proof. The simulation is constructed as follows:

1) For generating a vector of Boolean sharings $\langle r\rangle^b$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{E2A}}$ by recording the shares of corrupted parties on $\langle r\rangle^b$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{E2A}}$.
2) If the table is private, for $i \in \mathcal{H}$, $\mathcal{S}$ sends a fresh zero-ciphertext $[\![\mathbf{0}]\!]$ to $\mathcal{A}$, and initializes $[\![\boldsymbol{m}]\!] := [\![\mathbf{0}]\!]$ if P_1 is honest. If the table is public and P_1 is honest, $\mathcal{S}$ initializes $[\![\boldsymbol{m}]\!]$ following the protocol specification.
3) For each $i \in \mathcal{H}$, $\mathcal{S}$ generates a fresh zero-ciphertext $[\![\mathbf{0}]\!]$ and sends it to $\mathcal{A}$.
4) For each arithmetic sharing $\langle T_i'\rangle^a$ with $i \in [0, N/2-1]$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{E2A}}$ by recording the corrupted parties' shares sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{E2A}}$.
5) For each $i \in [0, N/2-1]$, $\mathcal{S}$ sends the shares of corrupted parties about $\langle r_i\rangle^b$ and $\langle T_i'\rangle^a$ to functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$.

It is easy to see that the simulation of $\mathcal{F}_{\mathsf{E2A}}$ is perfect. Under the assumption that the THE scheme is CPA secure, the simulation in the above step (2) using a zero ciphertext is computationally indistinguishable from the real ciphertexts. Furthermore, under the assumption that the THE scheme satisfies circuit privacy, the simulation in the above step (3) using a zero ciphertext is also computationally indistinguishable from the real ciphertext. In both worlds, the output shares of corrupted parties and honest parties satisfy that their sum is the correct value. Therefore, the joint distributions of the adversary's view and the honest parties' outputs are computationally indistinguishable in both worlds. □

B.2. Proof of Theorem 2

Proof. The simulation is constructed as follows:

1) For generating a vector of Boolean sharings $\langle r\rangle^b$ with $r \in \{0,1\}^m$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{E2A}}$ by recording the shares of corrupted parties about $\langle r\rangle^b$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{E2A}}$.
2) If the table is private, for $i \in \mathcal{H}$, $\mathcal{S}$ sends a fresh zero-ciphertext $[\![\mathbf{0}]\!]$ to $\mathcal{A}$, and initializes $[\![\boldsymbol{m}]\!] := [\![\mathbf{0}]\!]$ if P_1 is honest. If the table is public and P_1 is honest, $\mathcal{S}$ initializes $[\![\boldsymbol{m}]\!]$ following the protocol description.
3) For each $i \in \mathcal{H}$, $\mathcal{S}$ generates a fresh zero-ciphertext $[\![\mathbf{0}]\!]$ and sends it to $\mathcal{A}$.
4) For generating arithmetic sharing $\langle T'\rangle^a$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{E2A}}$ by recording the shares of corrupted parties on $\langle T'\rangle^a$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{E2A}}$.
5) $\mathcal{S}$ sends the shares of corrupted parties about $\langle r\rangle^b$ and $\langle T'\rangle^a$ to functionality $\mathcal{F}_{\mathsf{Prep-LUT}}$.

The simulation of $\mathcal{F}_{\mathsf{E2A}}$ is perfect. Under the assumption that the THE scheme is CPA secure and satisfies circuit privacy, the simulation using zero ciphertexts is computationally indistinguishable from the real ciphertexts. In both the ideal-world execution and real-world execution, the output shares of corrupted parties and honest parties satisfy the correct correlation on additive sharings. □

B.3. Proof of Theorem 3

Proof. The simulation is constructed as follows:

1) $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Prep-LUT}}$ by recording the shares of corrupted parties w.r.t. $\langle r\rangle^b$ and $\langle T'\rangle^a$ received from $\mathcal{A}$.
2) Given the shares of corrupted parties on input sharings $\langle x\rangle^b$, $\mathcal{S}$ computes the shares of corrupted parties w.r.t. $\langle u\rangle^b = \langle x\rangle^b \oplus \langle r\rangle^b$.
3) $\mathcal{S}$ samples $u \leftarrow \{0,1\}^m$, and then samples the shares of honest parties uniformly such that the sum of the shares of all parties is equal to u. Then, $\mathcal{S}$ sends the shares of honest parties to $\mathcal{A}$ during the $\mathsf{Open}(\langle u\rangle^b)$ procedure.
4) $\mathcal{S}$ sends the shares of corrupted parties about $\langle T[x]\rangle^a$ to functionality $\mathcal{F}_{\mathsf{LUT}}$.

Clearly, the simulation of $\mathcal{F}_{\mathsf{Prep-LUT}}$ is perfect. In the real protocol execution, u is uniform in $\{0,1\}^m$ due to the uniformity of r. Therefore, the simulation of string u is also perfect. In both worlds, the output shares of corrupted parties and honest parties satisfy the correct correlation about arithmetic sharings. Overall, the joint distributions of the adversary's view and the honest parties' outputs are perfectly indistinguishable in both worlds. □

B.4. Proof of Theorem 5

Proof. The simulation is constructed as follows:

1) $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{LUT}}$ by recording the shares of corrupted parties on Boolean sharings $\langle \boldsymbol{r}\rangle^b$ received from $\mathcal{A}$.
2) $\mathcal{S}$ invokes the simulator for Π_{B2A} to simulate the generation of arithmetic sharing $\langle r\rangle^a$. During the procedure, $\mathcal{S}$ records the shares of corrupted parties on $\langle r\rangle^a$.
3) If $(2^\ell - p)/2^\ell > 1/2^\rho$, then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{LUT}}$ by sampling $\boldsymbol{r}' \leftarrow \{0,1\}^\ell$ and outputting the bit that indicates if $\sum_{j\in[0,\ell-1} 2^j \cdot \boldsymbol{r}'[j] < p$ to $\mathcal{A}$. If $\sum_{j\in[0,\ell-1} 2^j \cdot \boldsymbol{r}'[j] \geq p$, $\mathcal{S}$ restarts the protocol simulation.
4) Given the shares of corrupted parties on $\langle x\rangle^a$, $\mathcal{S}$ computes the corrupted parties' shares about $\langle u\rangle^a = \langle x\rangle^a - \langle r\rangle^a$. $\mathcal{S}$ samples $u \leftarrow \mathbb{Z}_p$ and the shares of honest parties uniformly such that the shares of all parties sum to u. Then, $\mathcal{S}$ sends u to $\mathcal{A}$.
5) $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{LUT}}$ by recording the shares of corrupted parties about $\langle \boldsymbol{x}\rangle^b$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{LUT}}$. Then, $\mathcal{S}$ sends these shares to functionality $\mathcal{F}_{\mathsf{Conv}}$.

It is easy to see that the simulation of $\mathcal{F}_{\mathsf{LUT}}$ is perfect. Following the proof of Theorem 4, the simulation of sub-protocol Π_{B2A} is also perfect. If $(2^\ell - p)/2^\ell > 1/2^\rho$, through the "rejection-sampling" procedure, we guarantee that $r = \sum_{j\in[0,\ell-1} 2^j \cdot \boldsymbol{r}[j]$ is uniform in $\mathbb{Z}_p$. Otherwise, the distribution of r is identical to the uniform distribution in $\mathbb{Z}_p$, except with probability at most $1/2^\rho$. For the simulation checking if $r < p$, $\mathcal{S}$ samples a random string $\boldsymbol{r}'$ that has the same distribution as the real string $\boldsymbol{r}$, and simulates the output bit by deciding if $\sum_{j\in[0,\ell-1} 2^j \cdot \boldsymbol{r}'[j] < p$. Clearly, this is perfectly indistinguishable from the real protocol execution. From the uniformity of $r \in \mathbb{Z}_p$, we have that u is uniform in $\mathbb{Z}_p$ except with probability at most $1/2^\rho$ in the real protocol execution. Therefore, the simulation of u is statistically indistinguishable from the real protocol execution. In both the real-world execution and ideal-world execution, the output shares of all parties satisfy the correct correlation about arithmetic sharings. In conclusion, the joint distributions of the adversary's view and the honest parties' outputs are statistically indistinguishable in both worlds. □

B.5. Proof of Theorem 6

Proof. The simulation is constructed as follows:

1) For each wire w, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{LUT}}$ by recording the shares of corrupted parties on $\langle\lambda_w\rangle^b$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{LUT}}$.
2) For each AND gate g with input wires u, v, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{LUT}}$ by recording the shares of corrupted parties about $\langle\lambda_u\lambda_v\rangle^b$ received from $\mathcal{A}$. Following the protocol specification, for each gate g with output wire w, $\mathcal{S}$ computes the corrupted parties' shares on $\langle e_{w,\alpha,\beta}\rangle^b$ for each $\alpha, \beta \in \{0, 1\}$.
3) For the output wire w of each gate g, $\mathcal{S}$ simulates the generation of $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$ for each $\alpha, \beta \in \{0, 1\}$.
   - If g is an AND gate, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{LUT}}$ by receiving the corrupted parties' shares on $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$ from $\mathcal{A}$.
   - If g is a XOR gate, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Prep-LUT}}$ by recording the shares of corrupted parties on $\langle T'\rangle^a$ and $\langle r\rangle^b$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Prep-LUT}}$. Then, $\mathcal{S}$ computes the corrupted parties' shares about $\langle u\rangle^b = \langle e_{w,0,0}\rangle^b \oplus \langle r\rangle^b$. $\mathcal{S}$ samples $u \leftarrow \{0, 1\}$ and the shares of honest parties such that the shares of all parties are sum to u. For $\mathsf{Open}(\langle u\rangle^b)$, $\mathcal{S}$ sends the shares of honest parties on $\langle u\rangle^b$ to $\mathcal{A}$. Following the protocol specification, $\mathcal{S}$ computes the shares of corrupted parties on $\langle \boldsymbol{k}_{w,e_{w,\alpha,\beta}}\rangle^a$.
4) For generating garbled rows, $\mathcal{S}$ simulates as follows:
   a) For each wire w, $\mathcal{S}$ samples a masked value $e_w \leftarrow \{0, 1\}$ at random. Thus, for each gate g with input wires u, v, $\mathcal{S}$ knows masked values $e_u, e_v \in \{0, 1\}$.
   b) $\mathcal{S}$ can obtain the "active path" indicating which garbled rows $\mathcal{A}$ can decrypt. That is, for each gate g with input wires u, v, $\mathcal{A}$ can only decrypt the garbled row indexed by (g, e_u, e_v).
   c) For each wire w, following the protocol specification, $\mathcal{S}$ generates $\boldsymbol{k}^i_{w,e_w} \leftarrow \mathsf{SecKeyGen}(pp)$ for $i \in \mathcal{H}$.
   d) For each gate g with input wires u, v and output wire w, for each $\alpha, \beta \in \{0, 1\}$ and $i \in \mathcal{H}$, $\mathcal{S}$ sets $[\![\boldsymbol{k}^i_{w,e_{w,\alpha,\beta}}]\!] = [\![\boldsymbol{0}]\!]$ if $(\alpha, \beta) \neq (e_u, e_v)$ and computes $[\![\boldsymbol{k}^i_{w,e_{w,e_u,e_v}}]\!] \leftarrow \mathsf{Enc}_{\boldsymbol{k}^i_{u,e_u}+\boldsymbol{k}^i_{v,e_v}}((g, e_u, e_v), \boldsymbol{k}^i_{w,e_w})$, where $[\![\boldsymbol{0}]\!]$ is a fresh zero ciphertext. Then, for $i \in \mathcal{H}$, $\mathcal{S}$ sends $[\![\boldsymbol{k}^i_{w,e_{w,\alpha,\beta}}]\!]$ for each $\alpha, \beta \in \{0, 1\}$ to $\mathcal{A}$.
   e) If P_1 is honest, $\mathcal{S}$ receives the AHE ciphertexts from $\mathcal{A}$ for each corrupted party $P_i \in \mathcal{M}$.
5) Through the above step, $\mathcal{S}$ simulates a garbled circuit $\mathcal{GC}$. In the online phase, $\mathcal{S}$ holds the corrupted parties' shares on all input bits.
6) For each circuit-input wire $w \in \mathcal{I}$, $\mathcal{S}$ computes the shares of corrupted parties about $\langle e_w\rangle^b = \langle x_w\rangle^b \oplus \langle\lambda_w\rangle^b$, and samples the honest parties' shares on $\langle e_w\rangle^b$ such that the shares of all parties sum to e_w chosen by itself. $\mathcal{S}$ simulates the $\mathsf{Open}(\langle e_w\rangle^b)$ procedure by sending the shares of honest parties to $\mathcal{A}$.
7) For each circuit-input wire $w \in \mathcal{I}$, on behalf of every honest party P_i, $\mathcal{S}$ sends $\boldsymbol{k}^i_{w,e_w}$ to $\mathcal{A}$. If P_1 is honest, $\mathcal{S}$ also receives the keys of corrupted parties from $\mathcal{A}$.
8) For each circuit-output wire $w \in \mathcal{O}$, for each corrupted party $P_i \in \mathcal{M}$, $\mathcal{S}$ sets its share on $\langle y_w\rangle^b$ as $\langle\lambda_w\rangle^b_i$ if $i \neq 1$, or computes its share on $\langle y_w\rangle^b$ by $e_w \oplus \langle\lambda_w\rangle^b_1$ otherwise. Then, $\mathcal{S}$ sends the shares of corrupted parties on $\langle y_w\rangle^b$ to functionality $\mathcal{F}_{\mathsf{MPC}}$.

It is clear that the simulation of $\mathcal{F}_{\mathsf{LUT}}$ and $\mathcal{F}_{\mathsf{Prep-LUT}}$ is perfect. For the output wire w of each XOR gate, a bit $u = e_{w,0,0} \oplus r$ is opened in the real protocol execution. Since r is a uniform bit, $u \in \{0, 1\}$ is random. In the ideal-world execution, $\mathcal{S}$ directly samples a random bit u and opens it by sending the shares of honest parties. Therefore, the distribution of u is the same in both worlds. Furthermore, the shares of honest parties on $\langle u\rangle^b$ are uniform under the condition that the sum of all shares is equal to u in both worlds. Hence, these shares sent by $\mathcal{S}$ during the $\mathsf{Open}(\langle u\rangle^b)$ procedure have the identical distribution in both worlds.

As for the simulation of garbled circuits, we first consider the case that P_1 is corrupted. The evaluator P_1 knows all the keys and masked values in the active path, and thus is able to decrypt all the AHE ciphertexts received from every honest party P_i in the active path in the real protocol execution. In the ideal-world execution, these AHE ciphertexts sent by every honest party P_i are generated honestly by $\mathcal{S}$ following the protocol specification. Therefore, the AHE ciphertexts of every honest party in the active path have the identical distribution in both worlds. Differently, the AHE ciphertexts sent by every honest party outside the active path encrypt the corresponding keys in the real protocol execution, while these ciphertexts encrypt the zero vector in the ideal-world execution. We can bound the difference in two worlds by a hybrid argument based on the assumption that the private-key AHE scheme is CPA secure. If P_1 is honest, the analysis is the same, where $\mathcal{A}$ learns less information in this case.

In the online phase of the real protocol execution, the opened bit e_w for each circuit-input wire w is the XOR of the real input bit x_w and random mask λ_w. We first consider the case that P_1 is corrupted. We note that $e_{w,\alpha,\beta}$ for the output wire w of each gate and $\alpha, \beta \in \{0, 1\}$ except for the opened bit $e_w = e_{w,e_u,e_v}$ are kept secret under the assumption that the private-key AHE scheme is CPA secure. Therefore, in the real protocol execution, λ_w for each wire w is a uniform bit, which guarantees that e_w is random in $\{0, 1\}$. In the ideal-world execution, $e_w \in \{0, 1\}$ is sampled at random by $\mathcal{S}$. Furthermore, the keys sent in the online phase have the same distribution in both worlds. If P_1 is honest, then $\mathcal{A}$ learns less information. Overall, for both cases, the distributions of e_w and the keys are computationally indistinguishable in both worlds. In both the real-world execution and ideal-world execution, the shares of all parties on circuit-output bits satisfy the correct correlation of Boolean sharings. In conclusion, the joint distributions of the adversary's view and the honest parties' outputs are computationally indistinguishable in both worlds. □

Appendix C.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

The key contribution of this paper is the construction of efficient MPC protocols for performing secret lookups into public and private tables. The authors use these tables to obtain highly-efficient implementations of two key applications: (1) Protocols for conversion to and from boolean shares to arithmetic shares. Such protocols are a key component of MPC protocols that aim to efficiently support boolean and arithmetic computations (e.g., ML computations), and the new protocols enable orders-of-magnitude speedups for these protocols. (2) Protocols for multiparty garbled circuits, which achieves asymptotically optimal communication costs, and results in concrete improvements in communication overhead.

The authors implement the core subprotocols, provide microbenchmarks of these, and estimate the costs of various mixed-mode computations when using these protocols.

C.2. Scientific Contributions

- Addresses a long-known issue in MPC protocols
- Provides a valuable contribution

C.3. Reasons for Acceptance

1) Most real-world computations contain a mix of both bitwise computations (ANDs, XORs, etc.) and arithmetic computations (ADDs, MULs, etc). However, most existing MPC protocols cannot efficiently support both workloads simultaneously, i.e. they either support bitwise computations, or arithmetic computations. This paper provides a highly efficient method for switching between these two kinds of MPC protocols, thus enabling efficient execution of mixed-workload computations.
2) The paper is generally well-written: it motivates the problem well, and explains its techniques clearly.
3) The key underlying technique is elegant, and lends itself to a clean and simple implementation.

C.4. Noteworthy Concerns

1) While the overall communication of the proposed garbling scheme is linear in the number of parties, it comes at the expense of requiring communication for garbling XOR gates as well. This is a major downside considering that all the size optimized circuits have a huge discrepancy in the number of AND and XOR gates (#XOR $>>$ #AND) and the existing methods can garble XOR gates for free.
2) The scheme is limited to semi-honest security and achieving malicious security is non-trivial.