AutoHoG: Automating Homomorphic Gate Design for Large-Scale Logic Circuit Evaluation

Zhenyu Guan, *Member, IEEE,* Ran Mao, Qianyun Zhang, *Member, IEEE,* Zhou Zhang, Zian Zhao, and Song Bian, *Member, IEEE*

***Abstract*—Recently, an emerging branch of research in the field of fully homomorphic encryption (FHE) attracts growing attention, where optimizations are carried out in developing fast and efficient homomorphic logic circuits. While existing works have pointed out that compound homomorphic gates can be constructed without incurring significant computational overheads, the exact theory and mechanism of homomorphic gate design have not yet been explored. In this work, we propose AutoHoG, an automated procedure for the generation of compound gates over FHE. We show that by formalizing the gate generation procedure, we can adopt a match-and-replace strategy to significantly improve the evaluation speed of logic circuits over FHE. In the experiment, we first show the effectiveness of AutoHoG through a set of benchmark gates. We then apply AutoHoG to optimize common Boolean tasks, including adders, multipliers, the ISCAS'85 benchmark circuits and the ISCAS'89 benchmark circuits. We show that for various circuit benchmarks, we can achieve up to $5.7\times$ reduction in computational latency when compared to the state-of-the-art implementations of logic circuits using conventional gates.**

***Index Terms*—homomorphic encryption, compound gate design, circuit synthesis, logic replacement**

I. Introduction

COMPUTING over homomorphically encrypted data is drawing increasing attention across different academic disciplines, ranging from cryptography, security, and machine learning to hardware design and electronic design automation (EDA). The main reason that the EDA community is involved in the development of fully homomorphic encryption (FHE) is because of the fact that computing over FHE is fundamentally similar to that over electronic devices. Therefore, design philosophies devised for integrated circuits can easily be ported to formulate new methods of computing over FHE ciphertexts [1]–[3]. In particular, we see rising interest in the design and optimization of logic circuits over FHE [3], [4], since non-linear operations are the main bottleneck in FHE applications [5], [6].

We thank the anonymous reviewers and shepherds for their helpful feedback.

This work was partially supported by the National Key R&D Program of China (2023YFB3106200), the National Natural Science Foundation of China (62002006, 62172025, U21B2021, 61932011, 61932014, 61972018, 61972019, 62202028, U2241213). This work is also supported by Huawei Technologies Co., Ltd.

Corresponding author is Song Bian.

Z. Guan, R. Mao, Q. Zhang, Z. Zhang, Z. Zhao and S. Bian are with School of Cyber Science and Technology, Beihang University, Beijing 100191, China (e-mail: guanzhenyu@buaa.edu.cn; maoran_44@buaa.edu.cn; zhangqianyun@buaa.edu.cn; zhouzhang@buaa.edu.cn; zhaozian@buaa.edu.cn; sbian@buaa.edu.cn).

The main motivation behind the design of logic circuits stems from the fact that logic operations are intrinsically slower than arithmetic tasks. Here, we take the comparison between the amount of 16-bit multiply-accumulate [7] (MAC) (a typical type of arithmetic operation) and single-bit AND gates [8] (a clear logic operation) that can be homomorphically evaluated within a second as an example. While the computational complexity of MAC is at least $16\times$ (on the basis of bit width) than that of 1-bit AND, using the-state-of-the-art implementations, we can achieve a running speed of homomorphic MAC that is more than $22{,}000\times$ higher than that of homomorphic AND. As it turns out, the cost of the homomorphic evaluation of *polynomial* and *non-polynomial* circuits are extremely imbalanced.

To narrow the gap between polynomial and non-polynomial homomorphic evaluations, there are two existing approaches: one is to use polynomial approximation [6], [9], and the other is based on designated logic circuit [10]. While the polynomial approximation can attain sufficient accuracy with lower computational costs in certain applications [6], the approximation error renders such an approach impractical for high-precision logic circuits which can be widely found in HE applications [11]–[13]. Meanwhile, the other lane of research investigates how to improve the efficiency of logic evaluation, notably works related to the torus FHE (TFHE) scheme [3]–[5], [10], [14], [15]. In particular, it was first shown in [4] that certain compound logic gates (e.g., a full adder) can be homomorphically evaluated at the same computational cost as standard cells (e.g., an AND). Consequently, we can reduce the overall costs of logic circuit evaluation by substituting concatenated standard cells with low-cost compound homomorphic gates.

Unfortunately, the gate derivation in [4] appears to be rather empirical. We point out two important questions that were left unanswered in previous works on homomorphic logic circuits. First, it is not known if the compound gates and non-linear functions derived in [4] and [8] represent the complete set of homomorphic logic elements that are computable under the TFHE scheme, especially under parameter variation. Second, most existing works only provide decryptability analyses for TFHE ciphertexts under certain homomorphic evaluations. In other words, while we know that we can construct certain compound gates under a given the TFHE parameter set, we do not have formal noise analyses of why such gates are permitted, and if the same analyses apply to other types of compound homomorphic gates. Consequently, we are much in

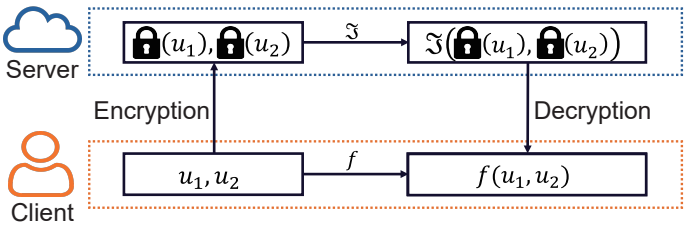

Fig. 1. The protocol illustration for a round of homomorphic evaluation. Here, the client first encrypts certain inputs and sends them to the server. The server then performs operations on the ciphertext and sends the modified ciphertext back to the client. Upon decryption, the client gets the same the result as if the execution is performed locally on its plaintext inputs.

need of concrete theories and design methodologies for FHE-based Boolean circuit designs.

In this work, we propose AutoHoG, an automated algorithm for the generation of homomorphic compound logic gates from a predefined set of FHE parameters. The insights and contributions of this work are summarized as follows.

- **Automatic Gate Generation**: We establish a general procedure for automatically generating compound logic gates over FHE schemes. Using such a procedure, we develop the AutoHoG framework to automatically replace simple gates with the generated compound gates.
- **Gate Decryptability Formulation**: To enable the above automatic gate generation procedure, we provide a theoretical formulation of the noise growth characteristics for the derived compound logic gates.
- **Applying to Complex Logic Circuits**: In the experiment, we show that, using the generated gates, the evaluation latency of homomorphic circuits can be reduced by as much as $5.7\times$ compared to the state-of-the-art methods. Our code is publicly available[1].

The rest of this paper is organized as follows. First, in Section II, we introduce preliminaries on FHE over logic circuits. Second, the AutoHoG framework is outlined in Sections III, where we detail the exact procedures for the automatic gate generation algorithms and gate decryptability derivations. Next, examples of generated gates along with circuit performance are presented in Section IV. Finally, we conclude our work in Section V.

II. Preliminaries and Related Works

A. Notations

Similar to previous works on FHE over the learning with errors (LWE) problem [16], [17], we use integers p to denote the plaintext modulus and integer q the ciphertext modulus. Integer n refers to the lattice dimension of integer LWE ciphertexts, and N is the lattice dimension of RLWE ciphertexts.

Besides, we note that a sequence of gates (i.e., a Boolean circuit) can be represented by a directed acyclic graph (DAG), where a cut refers to a set of graph nodes that can be mapped onto some Look-Up Tables (LUTs) [18]. In this work, we use the terms cut and sequence of gates interchangeably.

[1]https://github.com/Lavendes/AutoHog

B. Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is an advanced cryptographic scheme that allows arbitrary computations to be performed directly on encrypted data without the need of transferring secret keys. Almost all known FHE schemes are based on hard lattice problems [19], [20], mostly on the (variants of) LWE problem [16]. Lattice-based FHE enjoys a high level of security, as cryptographic primitives built over LWE are known to have worst-case to average-case reductions [21], [22] and are secure against quantum attacks [16], [17].

As illustrated in Fig. 1, a typical protocol based on FHE starts with the client sending some encrypted inputs to the server for further processing. The guarantee here is that computations over ciphertexts generate encrypted outcomes that, when decrypted, are the same as the results produced by a set of desired operations executed on the plaintexts. FHE has critical applications in various areas, particularly in scenarios involving sensitive information processed on untrusted servers, such as secure computation outsourcing [13] and privacy-preserving machine learning [5], [7], [23].

Despite numerous optimization strategies [23]–[27], homomorphic encryption still faces key challenges hindering its practical use. In particular, the computational complexity and ciphertext expansion limit the deployment of FHE in delay-sensitive or bandwidth-limited applications.

C. Bootstrapping and Types of FHE

Bootstrapping [19] is required for LWE-based HE schemes to actually be fully homomorphic. Specifically, for a ciphertext encrypting some plaintext message, applying bootstrapping on the ciphertext generates another ciphertext that encrypts the same plaintext message with a reduced ciphertext noise level. Bootstrapping is known to be a very costly operation in terms of latency and memory consumption.

Based on the differences in the exact bootstrapping process, we can divide existing FHE schemes [9], [20], [28]–[32] into two subcategories: arithmetic FHE and logic FHE. Here, we take CKKS [9] and TFHE [29] as representative examples of arithmetic and logic FHE schemes, respectively. In CKKS, bootstrapping is done through polynomial approximation. Here, the ciphertext modulus is first raised to a relatively large size (e.g., $\geq$ 600 bits). Then, an approximate sine function is evaluated, such that the level of the noise can be reduced [33]. While CKKS bootstrapping can be efficient when batching a large number of plaintext messages, the per-bootstrapping latency is relatively high [34]. For example, using the CKKS bootstrapping technique [35], it takes a CPU 903 seconds to precisely bootstrap a 420-bit message, which translates to an average bootstrapping speed of 2.15-bit per second.

Differing from the arithmetic approach, TFHE-like logic FHE directly applies non-linear operations on the FHE ciphertexts using techniques such as blind rotation [29]. The process of blind rotation is to homomorphically rotate an encrypted vector by a certain amount, such that the zeroth element in the decrypted vector corresponds to the original plaintext message. As a result, logic FHE not only enjoys lower

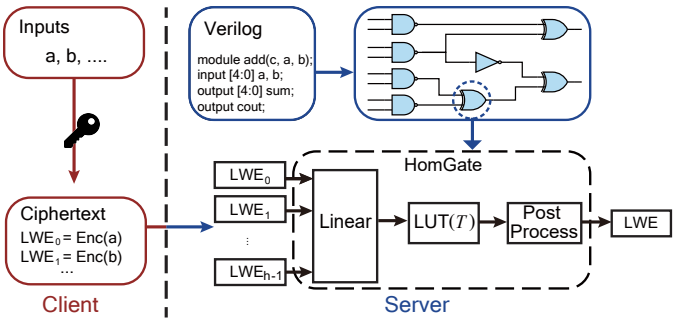

Fig. 2. A conceptual illustration for the workflow of the two-party homomorphic evaluation over logic gates as originally proposed in [29]. The evaluation of a homomorphic logic gate consists primarily of three steps: i) Linear: a linear combination of the input ciphertext, ii) LUT: a LUT evaluation performed through the ciphertext, iii) Post Processing: Operations restoring the ciphertext parameters to the original state, including the key switch process. The entire evaluation process is conducted on the server side.

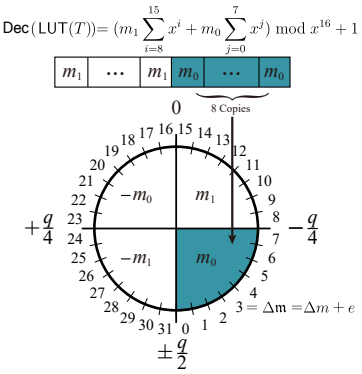

Fig. 3. The construction of $\mathsf{LUT}(T)$ that corresponds to the bootstrapping of the plaintext message m encrypted by LWE_m, where m can take two possible values, namely m_0 and m_1. Here, $T = \{m_0, m_1\}$ represents the identity function, $N = 16$. Dec is the decryption function. The $[N, 2N)$ part is the negative of $[0, N)$.

per-ciphertext bootstrapping latency [34], but is also highly flexible in applying non-linear logic functions [8]. For logic FHE [29], [30], the state-of-the-art implementation can evaluate bootstrapping at a speed of 62.5-bit per second [10], nearly 30× faster than arithmetic FHE for high-precision bootstrapping. Therefore, TFHE-like FHE schemes are generally preferred for homomorphic tasks that mainly evaluate logic circuits [5], [11]. While the parameterization and operator designs for arithmetic FHE have already become relatively well-understood [36], the potential in the capability of logic FHE is still under extensive research [3]–[5], [15].

As one of the important optimizations, homomorphic evaluation of a single-bit gate can be achieved through two additions for the input ciphertexts and a TFHE bootstrapping [29]. Specifically, two inputs and a constant offset are added together to determine the rotation degree during the blind rotation, while the output of the logic gate is encrypted as the vector to rotate (more details are discussed in Section II-D). [4] expanded this scheme to accommodate more compound gates, such as Full Adder and AOI21, by exploiting the additional degrees of freedom of the underlying linear relation. Nonetheless, [4] did not investigate the core mechanism for homomorphic gate generation, and it is not known if such gate designs are optimal. Therefore, the main objective of this work is to further explore the fundamental theories and practical applications of compound homomorphic gate designs.

D. Homomorphic Logic Gate Construction

Evaluation Procedure for Homomorphic Gate: Here, we first provide a high-level description of the homomorphic gate evaluation process, and explain some of the key operators later in this section. As abstractly shown in Fig. 2, the homomorphic gate evaluation mainly consists of three steps: i) linear combination Linear of a set of h input LWE ciphertexts, ii) the so-called LUT evaluation of the linearly combined result $\mathsf{LUT}(\mathsf{Linear}(\mathrm{LWE}_0, \cdots, \mathrm{LWE}_{h-1}))$, and iii) a post-processing stage that carries out the key switching procedure. In i), Linear computes a linear combination between the input LWE ciphertexts, i.e.,

$$\begin{aligned}&\mathsf{Linear}(\mathrm{LWE}_0, \cdots, \mathrm{LWE}_{h-1})\\ &= w_0 \cdot \mathrm{LWE}_0 + \cdots w_{h-1} \cdot \mathrm{LWE}_{h-1} = \sum_{i=0}^{h-1} w_i \cdot \mathrm{LWE}_i + \mathrm{offset},\end{aligned} \tag{1}$$

where $+$ is homomorphic addition and $\cdot$ is homomorphic constant multiplication. Here, both the set of weights $W = \{w_0, w_1, \cdots, w_{h-1}\}$ and offset are plaintext.

Since the post-processing stage is not directly related to the contributions of this work, we omit the details to simplify our presentation (more details can be found in [4], [29]).

LWE Ciphertext: An LWE ciphertext can be represented as $\mathrm{LWE}_m = (\mathbf{a}, b) \in_q^{n+1}$, an $n+1$ dimensional integer (modulo q) vector encrypting some plaintext message $m \in_p$. It holds that

$$b = (\mathbf{a} \cdot \mathbf{s} + \Delta m + e) \bmod q, \tag{2}$$

where $\mathbf{a} \in_q^n$ is a uniform random vector, $\mathbf{s} \in_{\{0,1\}}^n$ is a uniform random vector over the set $\{0, 1\}$, $\Delta \leq \lfloor q/p \rfloor \in$ is a scaling factor and e is some noise. Specifically, we define the decryption function as

$$\mathsf{Dec}(\mathrm{LWE}_m) = b - (\mathbf{a} \cdot \mathbf{s}) \bmod q = \Delta m + e. \tag{3}$$

Since e is sufficiently small, we can calculate $m = \lfloor \frac{\Delta m + e}{\Delta} \rceil$. We note that, in all LWE ciphertexts. e is sampled from a discrete Gaussian distribution with standard deviation σ, and is added during the fresh encryption process. Homomorphic function evaluation amplifies the noise level until the ciphertext becomes undecryptable. Therefore, as discussed in earlier works [2], [29] and further elaborated in Section III-D, automating noise estimation is essential in the design of homomorphic functions.

Lookup Table: As mentioned above, bootstrapping in TFHE-like FHE schemes [29], [37] is essentially the homomorphic rotation of an encrypted LUT, where the LUT is the encryption of some polynomial generated by table T. In

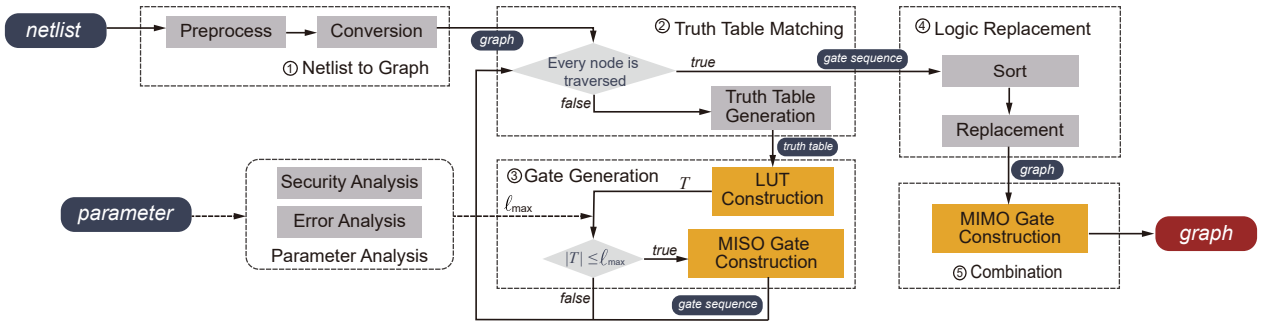

Fig. 4. An overview on the proposed AutoHoG framework with five steps: ① Netlist to Graph transforms a netlist into a directed acyclic graph, ② Truth Table Matching analyzes the graph by examining gate sequences and generating truth tables, ③ Gate Generation generates MISO compound gates based on the truth tables, ④ Logic Replacement replaces the sequences of gates that match with the compound gates generated in ③ and ⑤ Combination replaces multiple MISO gates with a single MIMO gate. The resulting graph represents a circuit that has been optimized to have a smaller size, as compared to the original circuit.

practice, we initialize the LUT encrypting some table $T = \{t_0, t_1, \cdots, t_{\ell-1}\}$ as the following degree-N polynomial

$$\mathsf{LUT}(T) = \mathsf{Enc}(\sum_{i=0}^{N-1} \tau_i x^i) = \mathsf{Enc}(\sum_{i=0}^{\ell-1} \sum_{j=0}^{N/\ell-1} t_i x^{j+N/\ell \cdot i}), \quad (4)$$

where x denotes the variable for the univariate polynomial, and T is a set of ℓ integers, and n is the lattice dimension specified in Section II-A.A conceptual illustration of the LUT rotation process is depicted in Fig. 3, where table T is equivalent to the identity function (i.e., $t_0 = 0$, $t_1 = 1$, and so on).

The input of the LUT is the ciphertext calculated by the Linear function, denoted as LWE_m. This ciphertext encrypts the plaintext message m. Let $\Delta \cdot \mathfrak{m} = \mathsf{Dec}(\mathrm{LWE}_m)$. The key idea of TFHE-like logic bootstrapping is first to fill $\mathsf{LUT}(T)$ with all possible values of m (in Fig. 3, m can either be m_0 or m_1). Then, we can homomorphically evaluate $x^{\Delta \mathfrak{m}}$, and compute $\mathsf{LUT}(T) \cdot x^{\Delta \mathfrak{m}}$. With an appropriate choice of parameters, we have that $\Delta \cdot \mathfrak{m} = \Delta \cdot m + \epsilon$ for some approximation error ϵ, and multiplying the $\mathsf{LUT}(T)$ with $x^{\Delta \mathfrak{m}}$ gives us

$$\mathsf{LUT}(T) \cdot x^{\mathsf{Enc}(\Delta \mathfrak{m})} = \mathsf{Enc}(\sum_{i=0}^{n-1} \tau_i x^i \cdot x^{\Delta m + e} \bmod x^n + 1) \quad (5)$$

$$= \mathsf{Enc}(\sum_{i=0}^{n-1} (\pm 1) \tau_{(i+\Delta m+e) \bmod 2N} \cdot x^i), \quad (6)$$

where Enc is the encryption function. Here, as long as we can make sure that $\tau_{0+\Delta \mathfrak{m}} = \Delta m$ when $\mathsf{LUT}(T)$ is generated, the zeroth coefficient in the shifted polynomial automatically encode the plaintext message Δm. Using Fig. 3 as an example, when $\Delta m = 3$ and the norm of the approximation error $||\epsilon|| \leq 3$, $\mathfrak{m} = \Delta m + \epsilon$ is within the range of 0 to 7. If we set all of the coefficients of $x^0, \cdots, x^7$ in the $\mathsf{LUT}(T)$ polynomial to be m_0, the rotation $\mathsf{LUT}(T) \cdot x^{\Delta \mathfrak{m}}$ will always have m_0 as the coefficient of the constant term.

E. LUT Technology Mapping

LUT mapping is an important step in field-programmable gate array (FPGA) design. A typical procedure for LUT mapping involves the following steps [38]: i) cut enumeration, ii) delay-optimum mapping, iii) area recovery and iv) writing out the resulting LUT network. Cut enumeration [38], [39] is a prevalent cut computation approach. For FPFAs with K-input LUTs, mapping involves computing K-feasible cuts for each two-input internal node. Unfortunately, this process becomes computationally intensive and storage intensive when the circuit sizes and K values become large. Some structural algorithms, like FlowMap [40] and CutMap [41], use the maximum-flow algorithm to find one optimal cut per node, reducing memory usage but increasing computational complexity and potentially mapping area. In contrast, some hardware synthesis tools, such as ABC [42], employ various optimization algorithms, including priority cut mapping [18], [43] and cut pruning [39], [44] to reduce the memory usage. Despite these efforts, practical constraints restrict the value of cut enumeration typically to be $K \leq 8$ due to computational and storage limitations. Second, in the delay-optimum mapping step, the computed cuts are sorted based on some specific order. Third, for area recovery, we first note that solving for an exact solution of the area minimization problem is categorized as NP- hard [45], rendering it intractable for larger circuits. Subsequently, various heuristics for approximate area minimization during mapping have been proposed, demonstrating promising results [46], [47]. Here, to reduce the mapping area, existing works [46] employ greedy strategies to iteratively modify the representative cuts of the nodes. Finally, the resulting network, where each and every node is covered by some LUT, is returned.

While LUT technology mapping shares similarity to the automated homomorphic gate generation problem, the difference lies in the fact that homomorphic compound gates can have a relatively large number of inputs when compared to semiconductor gates. Hence, for cuts with the number of inputs less than 5, the two problems are virtually equivalent. However, as sketched in Table III, homomorphic compound gates can have up to 32 inputs. As a result, the K-cut optimization algorithm needs to map cuts into a 32-input LUT, which requires a prohibitive amount of storage. Thus, it can be hard to directly apply conventional LUT technology mapping algorithms in optimizing homomorphic circuits.

III. AutoHoG: Automated Homomorphic Gate Generation

Here, we first provide an overview of each of the components in our framework in Section III-A. Next, the automatic construction methodology for multi-input single-output (MISO) and multi-input multi-output (MIMO) gates are explained in Section III-B and Section III-C, respectively. Finally, details on security analyses are presented in Section III-D.

A. Procedure Overview

An illustration of the AutoHoG procedure is provided in Fig. 4, which contains five main subcomponents: ① Netlist to Graph, ② Truth Table Matching, ③ Gate Generation, ④ Logic Replacement, and ⑤ Combination. First, ① Netlist to Graph converts a circuit netlist into a DAG, which is similar to that in conventional circuit synthesis flow. Second, ② Truth Table Matching traverses the graph and outputs one truth table that expresses the logic of a sequence of gates to be replaced. After receiving the truth table from ②, ③ Gate Generation tries to construct one multi-input single-output homomorphic compound gate that completely matches the above truth table while meeting the security and accuracy requirements. Then,④ Logic Replacement replaces the matching sequences of gates with the homomorphic compound gates generated in ③ in a decremental manner based on the number of gates replaced. Finally, the ⑤ Combination step reduces the circuit size by replacing the generated MISO compound gates with multi-input multi-output logic gates, producing a circuit DAG as the output of the overall AutoHoG framework. In what follows, we provide further details on each of the subcomponents.

① **Netlist to Graph**: First, we use the Netlist to Graph block to represent the circuit netlists as DAGs such that compound gates can be identified more easily. As mentioned above, the Netlist to Graph procedure is much similar to that in the conventional synthesis flow, where logic wires (i.e., input-output ports) are the nodes, and logic gates are the edges that connect the nodes. We numbered the nodes in a natural sequential order as they appear in the circuit graph.

② **Truth Table Matching**: The Truth Table Matching algorithm iterates over each node in the given DAG to identify the longest sequence of gates that can be represented by a single homomorphic logic gate. We first take the largest numbered node in the graph as the input, a gate sequence is created with the input node as its member. We employ a graph traversal algorithm to sequentially add the preceding nodes of the input node into the gate sequence. Every time a gate is added to the sequence, we generate the truth table associated with the gate sequence. Then, we proceed to Component ③ to check if such a truth table can be constructed using one MISO homomorphic compound gate. If Component ③ reports that such a gate cannot be constructed, or if the newly added nodes in the sequence are part of the circuit inputs, the algorithm aborts. Then, the gate sequence and the homomorphic gate constructed in the previous iteration by Component ③ is returned. The process is repeated until every node in the graph has been traversed.

③ **Gate Generation**: The Gate Generation block contains two sub-components: the parameter analysis and the gate construction steps. The parameter analysis is composed of error analysis and security analysis, where we check if a set of TFHE parameters meets the required decryption accuracy and security standard. The output of the parameter analysis step is $\ell_{\max}$, i.e., the maximum possible values $\mathsf{LUT}(T)$ can contain under such a parameter set (the maximum $|T|$). More details on the parameter analysis can be found in Section III-D. This step is a one-time operation performed during the setup phase of the proposed workflow. Then, upon receiving the truth table, we follow Algorithm 1 to execute the LUT Construction step, where we construct a look-up table T for the input truth table. The critical condition that T needs to be satisfied is that $|T| \leq \ell_{\max}$. If such condition is not met, we report to ② Truth Table Matching component and re-execute the matching process. More details on the construction of T and the homomorphic compound gates are available in III-B.

④ **Logic Replacement**: The Logic replacement block simply replaces the sequence of gates matched in ② by the MISO compound gate generated in ③. The replacement process begins with the sequence having the largest number of gates. Whenever a node is replaced, the corresponding gate sequence is emptied before the next sort. The replacement block generates a new DAG with the corresponding gates replaced.

⑤ **Combination**: With a fully replaced DAG in hand, we use Combination to further reduce the number of gates in the logic circuit. We point out that TFHE is capable of constructing multi-output gates, and we optimize the circuit by combining multiple MISO gates to form MIMO gates. Hence, we search for logic gates that have the same inputs in the final DAG given by ④. A more detailed explanation on the construction of MIMO gate can be found in Section III-C.

Complexity Analysis for AutoHoG: The time complexity of the AutoHoG framework can be expressed as:

$$\mathcal{T}(s) = \mathcal{K} \sum_{i=0}^{s-1} \mathcal{H}_i, \tag{7}$$

where s is the number of gates in the circuit, $\mathcal{K}$ is a constant related to the chosen security parameter, and $\mathcal{H}_i$ is the time required to generate the i-th gate, which will be further explained in Section III-B.

In most cases, the gate generation time $\mathcal{H}$ for each node can be considered as a constant. Additionally, since the circuit graph is traversed only once, the time complexity of AutoHoG can be expressed as $O(s)$. In other words, the time consumption of the AutoHoG framework is linear to the size of the circuit.

B. Multi-Input Single-Output Gate Construction

In this subsection, we explain the construction of a homomorphic logic gate in accordance with a given truth table.

Construction of Table T: The construction of table T is the key step in building a logic gate over FHE. As depicted in Fig. 3, T is basically a sequence of logic 0's and 1's. Every logic value in the sequence represents the output of a particular combination of inputs, i.e., a row in some given truth table.

TABLE I
TABLE T FOR LOGIC FUNCTION $Y = a \wedge b \vee c \vee (\neg d)$ WITH $\mathsf{Linear} = 2a + 2b + 4c + d + 9/2$

a	b	c	d	Linear	T	Y
$-1/2$	$-1/2$	$-1/2$	$-1/2$	0	1	1
$-1/2$	$-1/2$	$-1/2$	$1/2$	1	0	0
$-1/2$	$-1/2$	$1/2$	$-1/2$	4	1	1
$-1/2$	$-1/2$	$1/2$	$1/2$	5	1	1
$-1/2$	$1/2$	$-1/2$	$-1/2$	2	1	1
$-1/2$	$1/2$	$-1/2$	$1/2$	3	0	0
$-1/2$	$1/2$	$1/2$	$-1/2$	6	1	1
$-1/2$	$1/2$	$1/2$	$1/2$	7	1	1
$1/2$	$-1/2$	$-1/2$	$-1/2$	2	1	1
$1/2$	$-1/2$	$-1/2$	$1/2$	3	0	0
$1/2$	$-1/2$	$1/2$	$-1/2$	6	1	1
$1/2$	$-1/2$	$1/2$	$1/2$	7	1	1
$1/2$	$1/2$	$-1/2$	$-1/2$	4	1	1
$1/2$	$1/2$	$-1/2$	$1/2$	5	1	1
$1/2$	$1/2$	$1/2$	$-1/2$	8	1	1
$1/2$	$1/2$	$1/2$	$1/2$	9	1	1

Let F be the truth table, and assume that the i-th row in the truth table is $(X_i = \{x_{i,0}, \cdots, x_{i,h-1}\}, Y_i \in \{0,1\})$ where $F(X_i) = Y_i$. The main task in building T is then to find a group of weights $W = \{w\}$, such that, for all rows in T,

$$T[\mathsf{Linear}_W(x_{i,0}, \cdots, x_{i,h-1})] = T[\sum_{j=0}^{h-1} w_j x_{i,j}] = Y_i. \quad (8)$$

In other words, T represents a de-duplicated rearrangement of Y indexed by $\mathsf{Linear}_W(X)$. The objective is to establish the mapping $\mathsf{map} : \mathsf{Linear}_W(X) \rightarrow Y$, under the condition that

$$\mathsf{Linear}_W(X_i) = \mathsf{Linear}_W(X_j) \implies y_i = y_j \quad \forall i, j \in \{0, 1, \ldots, 2^h - 1\}. \quad (9)$$

Note that $\mathsf{Linear}_W(X_i) = W \cdot X_i$ is essentially an un-encrypted version of Eq. (1).

In Table I, we take a four input logic gate $Y = a \wedge b \vee c \vee (\neg d)$ as an example to better explain the above description. The input values are encoded from $\{0,1\}$ to $\{-\frac{1}{2}, \frac{1}{2}\}$. We set $W = \{2, 2, 4, 1\}$ with the constraint that a and b are considered equivalent within the scope of this logical function. Additionally, an offset of $\frac{9}{2}$ is utilized to guarantee the minimum value of the index of T is set to zero. Consequently, we derive T as $\{1, 0, 1, 0, 1, 1, 1, 1, 1, 1\}$, where each value in the set T corresponds to the output value of the truth table. Hence, if the outcome of Linear is 8 (i.e., $\{a, b, c, d\} = \{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, -\frac{1}{2}\}$), we can effortlessly locate the index 8 in T to obtain the output value 1, corresponding to $Y = 1 \wedge 1 \vee 1 \vee (\neg 0)$.

To accomplish the above objective, we derive a table construction procedure as given in Algorithm 1. First, on line 2–4 we set the weights of the inputs that are irrelevant to the output to be 0. If the truth table obtained by inverting a certain input is the same as the original truth table, we consider such inputs to be irrelevant to the output. Next, on line 5–9, we test for irrelevant inputs by exchanging the order of inputs in the truth table. If we can obtain an equivalent truth table even if the order of inputs is swapped, we consider such a pair of inputs to be equivalent. Here, we use Group_i to denote the i-th equivalent input group. Finally, on line 10–17, we identify the values of W that minimize $|T|$. Here, on line 10–11, the inputs from the same group are first set to have identical weight values, while inputs allocated to distinct groups are assigned with unique weight values. Then, on line 12–13, we calculate the total number of distinct groups, and generate the permutations over the set $\{0, 1, 2, 4, \ldots, 2^{h-1}\}$, representing the set of samples for W. Here, $\mathsf{Permutations}(S, g)$ computes the full permutation of g elements chosen from the set S. Lastly, on line 14–17, all samples in the set W_{sample} are tested, and the permutation that results in the minimum $|T|$ while meeting Eq. (9) is recorded. Ultimately, the algorithm concludes by returning the result with the minimum $|T|$.

Algorithm 1: LUT Construction Algorithm

Input : Truth table: $X = \{X_0, X_1, \ldots X_{h-1}\}; Y$
Output: Linear combination weights: $W = \{w_0, w_1, \ldots w_{h-1}\}; T$

```
1  Initialize Group
2  for i = 0 to h − 1 do
3      if Y[X_i] = Y[X̄_i] then
4          Add i to Group_0
5  for i = 0 to h − 1 do
6      if i not in any Group then
7          for j = i + 1 to h − 1 do
8              if Y[X_{i,j}] = Y[X_{j,i}] then
9                  Add j to Group_i
10 if i in Group_0 then w_i = 0;
11 if i, j in same Group then w_i = w_j;
12 g = number of distinct groups
13 W_sample ← Permutations({0, 1, 2, 4, ... 2^{h−1}}, g )
14 foreach W_temp in W_sample do
15     if Eq. (9) holds then T_temp[Linear_{W_temp}(X)] =Y;
16     if |T_temp| ≤ |T| then T = T_temp;W = W_temp;
17     return W, T
```

After identifying the distinct input groups, the task of constructing T can be equivalently transformed into an integer programming problem, expressed as:

$$\begin{aligned} &\text{Minimize} \quad |T| \\ &\text{Subject to} \\ &\quad Eq.\ (9) \text{ holds} \\ &\quad 0 \leq w_i \quad \forall i \in \{0, 1, \ldots, 2^{h-1}\}, \\ &\quad w_i = 0 \quad \forall i \in \mathrm{Group}_0, \\ &\quad w_i = w_j \quad \forall i, j \text{ in same Group}, \\ &\text{and} \\ &\quad W \in^h \end{aligned} \quad (10)$$

The conditions imposed by Eq. (9) and the requirements of values assigned to W serve as the constraints in the integer programming problem. The objective is to determine the optimal value of the integer vector W, which minimizes $|T|$. A typical non-linear solver can be used to generate an (approximate) optimal W. A subtle difference between Algorithm 1 and Eq. (10) is the range of w_i. In Algorithm 1, we sample weights from the set $\{0, 1, 2, 4 \ldots, 2^{h-1}\}$, while Eq. (10) samples weights from all the integers between 0 and 2^{h-1}. This more flexible

approach allows the solver to explore a broader set of W values, potentially leading to better outcomes.

Correctness Analysis for Gate Generation: Eq. (8) holds if the mapping between the Linear results and the truth table is satisfied. The mapping is achieved by appropriately setting the offset and weights for each input combination. The offset is set to make the smallest sum of products in Linear to be zero, while the weights differentiate the different inputs(i.e., X_i) and their corresponding truth table results. For a single input, the weight w_1 should be at least 1 to distinguish the two possible input value $\{-\frac{1}{2}, \frac{1}{2}\}$ and the $\text{offset} = -\min(\mathrm{w}_1 \times (-\frac{1}{2})) = \frac{1}{2}$. As a result, $\mathsf{Linear} = \{0, 1\}$. For two inputs, to differentiate not only the two possible values of a single input (i.e., $X_0 = 0$ or $X_0 = 1$) but also between the two different inputs (i.e., X_0 and X_1), w_2 needs to be equal to 2, while the offset is given by $-\min(w_1 + w_2 \times (-\frac{1}{2})) = \frac{3}{2}$. The output of Linear function is $\{0, 1, 2, 3\}$. If the two inputs are equivalent, i.e., in the same group as depicted in Algorithm 1, they can share the same weight since there is no need to distinguish between them. The parity argument suggests that for an h-input gate, a sequence of weights that satisfy the necessary conditions for the logic function to hold will always exist. Specifically, the sequence of weights will be of the form $\{w_0, \cdots, w_{h-1}\}$, where each $w_i \in \{0, 2^0, \cdots, 2^{h-1}\}$, and the linear mapping satisfies Eq. (8) with the interval for each output is 1.

Complexity Analysis for Gate Generation: The most time-consuming part of the algorithm is the weight sampling process on line 14–16. The loop iterates through all possible permutations of W and records T that satisfy Eq. (9). The result with the shortest length is determined through comparison. The input size of Algorithm 1 is denoted by h, which is the length of the inputs, and the group number is g. In the worst case, this process involves a full permutation of g elements from a set of h elements, resulting in a time complexity that is proportional to the total number of possible permutations, which is

$$\mathcal{H}(h, g) = O(\frac{h!}{(h-g)!}). \tag{11}$$

Despite the high level of computational complexity, in practice, h and g are typically small due to the encryption parameters satisfying the necessary security and accuracy requirements.

Negacyclic Property: The polynomial multiplication operation involved in the LUT rotation takes place under the modular arithmetic of the 2N-th cyclotomic polynomial. Therefore, it presents a negacyclic property [24], denoting that a polynomial p satisfies $p \cdot x^N = -p$. As depicted in Fig. 3, $\Delta\mathfrak{m}$ and $\Delta\mathfrak{m} + N$ are situated in opposite positions, suggesting an inverse correlation between their corresponding values. This characteristic is leveraged to achieve a smaller T. We search from the middle of the table backward until an ℓ is judged in accord with the set:

$$t_i = \neg\, t_{i+\ell} \quad (i = \{0, 1, \ldots, |T| - 1 - \ell\}, \frac{|T|}{2} \leq \ell \leq |T|) \tag{12}$$

The T is updated to $T = \{t_0, t_1, \ldots t_\ell\}$. The rest part is regarded as the $(N, 2N]$ coefficients of the polynomial, which equals the inversion of T.

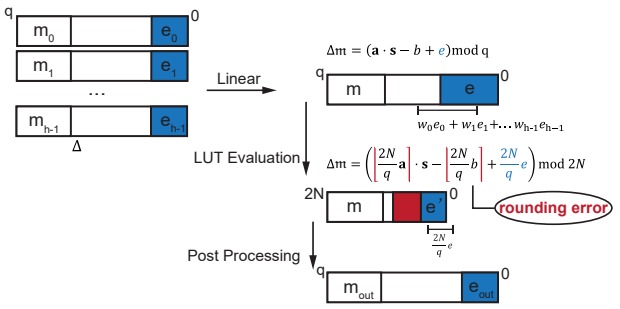

Fig. 5. Variation of noise in the homomorphic evaluation. The noise is a product of the given input noise (blue) and the rounding error inherent in the computations (red).

C. Multi-Input Multi-Output Gate Construction

Multi-value Bootstrapping: The idea of multi-value bootstrapping described in [48] is a technique to evaluate multiple LUTs at the cost of a single bootstrapping. We observe that such a technique can easily be integrated into our framework, such that multiple MISO gates can be merged to form one MIMO gate. Recall that, the rotation of $\mathsf{LUT}(T)$ can be expressed as

$$\mathsf{LUT}(T) \cdot x^{\mathsf{Enc}(\Delta\mathfrak{m})} = \mathsf{Enc}(\sum_{i=0}^{\ell-1} \sum_{j=0}^{N/\ell-1} t_i x^{j+N/\ell \cdot i} \cdot x^{\Delta\mathfrak{m}}) \tag{13}$$

$$= \mathsf{Enc}((\sum_{i=0}^{N/\ell-1} x^i \cdot x^{\Delta\mathfrak{m}}) \cdot \sum_{i=0}^{\ell-1} t_i x^{N/\ell \cdot i}). \tag{14}$$

The main observation is that $\sum_{i=0}^{N/\ell-1} x^i$ can be regarded as a common factor for different tables T. The LUT evaluation is divided into two parts: the expensive homomorphic rotation over the common factor, and the cheap multiplication with a polynomial constructed by a given T. To evaluate multiple LUTs, we can re-run Post Processing several times without repeating the LUT rotation step, which is the most time-consuming step in gate bootstrapping. In other words, in multi-value bootstrapping, we can generate multiple outputs at the cost of only one bootstrapping.

Single-Output Gate to Multi-Output Gate Combination: Using multi-value bootstrapping, we can combine multiple logic gates sharing the same inputs into one multi-output logic gate, where all gates share the same weight W. Hence, the objective here is to find a W so that multiple output mappings can be implemented simultaneously. Concretely, suppose we have k gates sharing the same inputs, where the truth table outputs are labeled as Y_j for $j \in \{0, \cdots, k-1\}$. Based on Algorithm 1, we first obtain k independent sets of MISO linear combination weights, namely $\{W_0, \cdots, W_{k-1}\}$ for each $\{Y_0, \cdots, Y_{k-1}\}$. Then, we can merge the gates if there exists a corresponding lookup table T_j such that at least one of the W_i satisfies

$$T_j[\mathsf{Linear}_{W_i}(X_i)] = Y_j, \tag{15}$$

for $j \neq i$. Note that we do not need Eq. (15) to hold for every $j \in \{0, \cdots, k-1\}$. However, the more gates that meet Eq. (15), the better acceleration ratios we can get. Eventually,

TABLE II
PARAMETER SETTING AND SECURITY IN DIFFERENT WORKS

Technique	LUT(T) $\log_2 q$	LUT(T) N	LWE $\log_2 q$	LWE n	λ (bit)
TFHE	32	1024	32	630	≥ 127
Remeo					
WAHC				636	
AutoHoG	64	2048	32	667	≥ 127

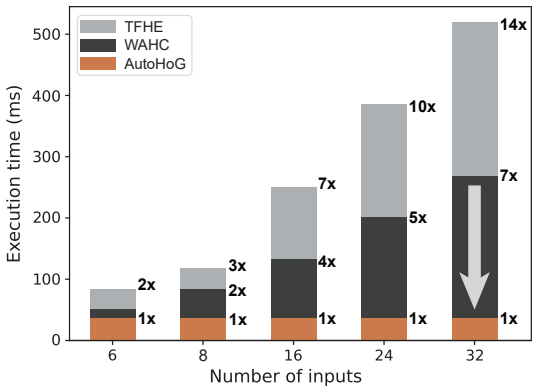

Fig. 6. Performance comparisons on a set of 5-, 6-, 16-, 24- and 32-input benchmark gates between AutoHoG and existing works. All tests considered are restricted to single input gate designs and allow for only one bootstrapping operation to construct the logic expressions. We are able to achieve a speedup of up to 7.4 $\times$.

a maximum of $k\times$ improvement in evaluation speed can be achieved if Eq. (15) holds for all $j \neq i$. Additionally, some small gates with different inputs can also be combined into MIMO gates. Their inputs are irrelevant and are combined as a new input to the MIMO gate.

D. Security Analyses

In this section, We first perform an evaluation of the security of the AutoHoG framework. Then we provide a theoretical analysis of the noise growth characteristics during the homomorphic gate evaluation, In particular, we show how such noise determines the size of the plaintext space, which in turn decides what kind of lookup tables T can be evaluated using a single homomorphic compound gate.

Security of AutoHoG: We point out that no additional security vulnerabilities are induced by the AutoHoG framework. The security of the AutoHoG procedure can be directly reduced to that of the underlying TFHE scheme, as the homomorphic evaluation of a logic gate is essentially a TFHE-like bootstrapping. The level of security is primarily influenced by the selection of the encryption parameters, such as the modulus and the ciphertext dimension, as well as the noise level in the ciphertexts. In general, larger parameter sizes and lower noise levels can provide stronger security assurances but also lead to increased computational complexity.

Gate Decryptability Analyses: As shown in Fig. 5, for an input LWE ciphertext, the ciphertext noise is mainly amplified by the Linear and LUT procedures. Given the modulus q, the scaling factor Δ, the LWE dimension n and the LUT(T) dimension N, we can derive the noise growth characteristic as follows.

$$\text{Error(output)} \approx \frac{2N}{q}\text{Error(input)} + \text{RoundingError}. \quad (16)$$

In the LUT procedure, there is a step where we map a ciphertext $\text{LWE} = (\mathbf{a}, b)$ from $_q$ to $_{2N}$. Let e be the initial noise contained in $(\mathbf{a}, b)$, this step re-scales the noise e to $e' = \frac{2N}{q}e$. In parallel, convert $(\mathbf{a}, b)$ to $(\lfloor \frac{2N}{q}\mathbf{a} \rceil, \lfloor \frac{2N}{q} b \rceil)$, which introduces a new noise caused by rounding. By simply adjusting the FHE encryption parameters (i.e., N and q), e' can be made small. Note that $\mathbf{a}$ is an n-dimensional vector, and each element introduces a rounding error. Thus, the RoundingError is, in fact, the accumulation of n rounding noises. As rounding errors follow a uniform distribution, the distribution of RoundingError can be described as the sum of n uniformly distributed random variable. Hence, the probability that RoundingError is greater than (or equal to) a certain threshold $\mathcal{E}$ can be expressed as:

$$P(\text{RoundingError} \geq \mathcal{E}) = 1 - \frac{1}{2^n r^n n!}\sum_{i=0}^{k}(-1)^i \binom{n}{i} (\mathcal{E} + nr - 2ir)^n, \quad (17)$$

where $r = \frac{q}{4N\times\Delta}$ and $k = \frac{rn+\mathcal{E}}{2r}$ [49]. Based on the maximum error rate that can be accepted, we can calculate $\mathcal{E}$. In order to get the correct result in LUT evaluation, each $2\mathcal{E}$ consecutive segments in LUT(T) should have the same logic value (e.g., $2\mathcal{E} = 8$ in Fig. 3). Consequently, we get $\ell_{\max} = \lfloor \frac{N}{2\mathcal{E}} \rfloor$

IV. EXPERIMENT

A. Experiment Setup

Here, we compare the circuit evaluation latency between AutoHoG and existing works, notably TFHE [29], WAHC [4] and Remeo [3]. We apply AutoHoG to a set of standard benchmark circuits, including adders, multipliers (i.e., C6288), the ISCAS'85 [50] benchmark circuits and the ISCAS'89 [51] benchmark circuits. The experiments for TFHE [29], WAHC [4] and AutoHoG are performed on a single core of the Intel Xeon Gold 6226R processor with 503 GBytes of RAM and the results of Romeo are directly taken from [3].

The parameters instantiated in AutoHoG and existing works are summarized in Table II. We doubled the dimension of LUT(T), and adjusted other parameters to reduce noise. This parameter is estimated to achieve 127-bit security by lwe-estimator [52], which is similar to existing works [3], [4], [29]. Although doubling dimensions N makes the running time of one bootstrapping expansion more than twice, using the replace strategy can still perform well in large-scale circuits.

B. Gate Benchmarks and Performance Comparisons

First, we summarize evaluation latency on a set of benchmark compound logic gates in Fig. 6. Since we can use only one bootstrapping to construct all of the tested MISO logic expressions, we can obtain as much as 7.4$\times$ speedup. From the TFHE parameters and gate decryptability analyses, we get

TABLE III
EXAMPLES OF HOMOMORPHIC BENCHMARK GATES WITH $\ell_{\max} = 32$

input number	weights	expression
≥33	[0,0,0...0,0,0]	$Y = 0; Y = 1$
32	[1,1,1...1,1,1]	$Y_1 = X_1 \wedge X_2 \wedge \cdots \wedge X_{31} \wedge X_{32}; Y_2 = X_1 \vee X_2 \vee ... \vee X_{31} \vee X_{32}$
24	[1,1,1...1,1,32]	$Y = X_1 \vee X_2 \vee \cdots \vee X_{23} \oplus X_{24}$
16	[1,2,2...,2,2,2]	$Y_1 = X_1 \wedge X_2 \oplus \cdots \oplus \neg((X_{16}); Y_2 = \neg(X_1) \vee X_2 \wedge \cdots \wedge X_{16}$
8	[1,1,1,1,8,8,8,8]	$Y_1 = \neg((X_1 \wedge \cdots \wedge X_4) \oplus (X_5 \vee \cdots \vee X_8); Y_2 = \neg((X_1 \vee \cdots \vee X_4) \oplus X_5 \oplus \cdots \oplus X_8$
6	[1,2,4,4,8,8]	$Y = X_1 \oplus (X_1 \vee X_2) \oplus (X_3 \wedge X_4) \cdots \oplus X_6$
	[1,1,4,4,4,4]	$Y = (X_1 \vee X_2) \wedge (X_3 \vee X_4 \vee X_5 \vee X_6)$
≤5		All 5,4,3,2,1 input gate

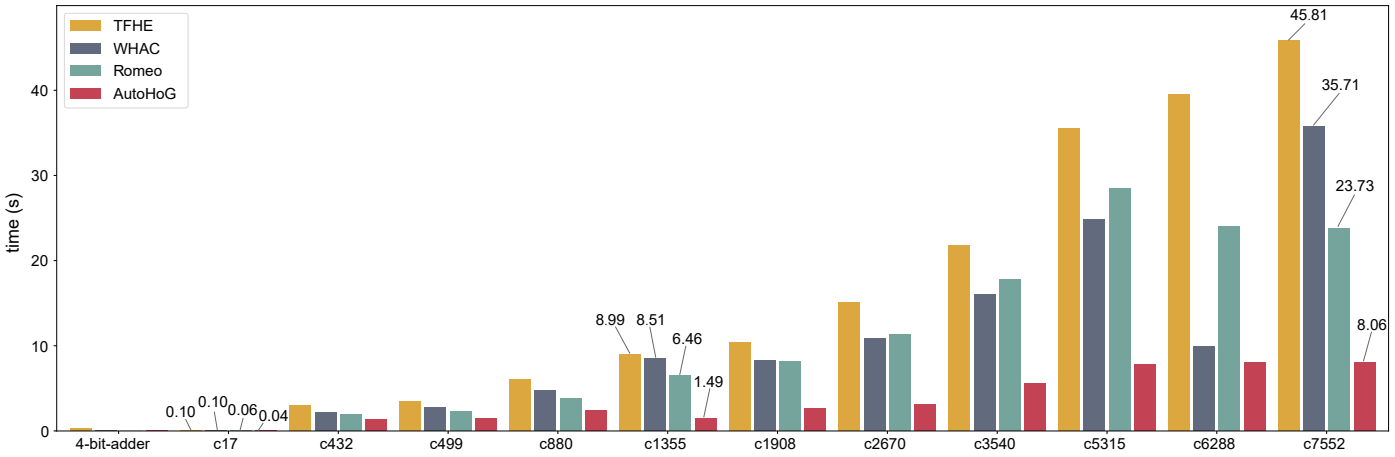

Fig. 7. Performance comparisons on ISCAS'85 benchmark circuits between AutoHoG and existing works. Our proposed approach has demonstrated a maximum improvement of up to 5.7×. While the actual acceleration ratios achieved may vary depending on the specific circuit topology, AutoHoG remains capable of outperforming most existing methods, as demonstrated by an average speedup of 2.8×.

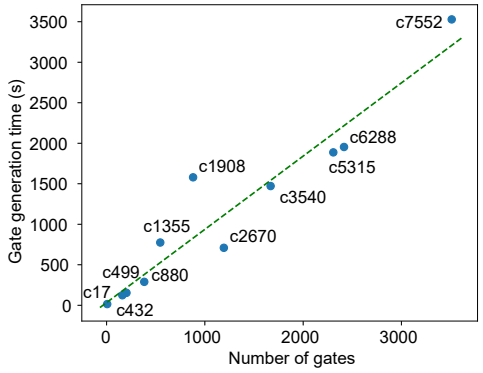

Fig. 8. Circuit replacement time of ISCAS'85 benchmark circuits. The horizontal axis represents the number of logic gates in the original circuit, while the vertical axis indicates the total time required to complete the replacement process after entering a netlist. As a general trend, the time consumption increases proportionally with the original circuit size.

$\ell_{\max} = 32$, i.e., we are able to construct logic gates with up to 32 inputs. However, the main reason that such large input gates can be evaluated is that the logic expressions contain many irrelevant or equivalent inputs.

Second, in order to effectively demonstrate the benchmark gates, we provide some specific logic expressions in Table III. First of all, we point out that in a tree of AND or OR gates, all inputs are equivalent. In such structures, all weights can be assigned to the value of 1, allowing for the evaluation of one compound gate with up to 32 inputs with only one gate bootstrapping. Meanwhile, the 8-input and 16-input gates give two expressions that share identical weights, enabling the construction of MIMO gates. Finally, our decryptability analyses in Section III-D show that one gate bootstrapping operation is sufficient to evaluate arbitrary gates with up to five inputs. We note that, when $\ell_{\max} = 32$, it is impossible to generate all gates with six or more inputs. For instance, consider the logic expression $Y_{\text{example}} = X_1 \wedge (\neg X_2) \oplus ((X_3) \vee (\neg X_4)) \oplus X_5 \vee (\neg X_6)$. This expression can only be represented with $W = \{1, 2, 4, 6, 8, 16, 32\}$, leading to $|T| = 61$. Even with the utilization of a solver, the minimum $|T|$ is 40, much larger than $\ell_{\max}$. As a result, Y_{example} cannot be expressed through a linear combination with the given parameter. Like Y_{example}, gates with six or more inputs involving few or no irrelevant inputs must have some large weights, leading to a $|T|$ larger than $l_{\max}$, which makes them impossible to be constructed.

Last but not least, we conducted a comparison experiment between Algorithm 1 and the Z3 SMT solver [53]. On average, the Z3 SMT solver achieves the same level of latency reduction compared to Algorithm 1, but requires much longer running

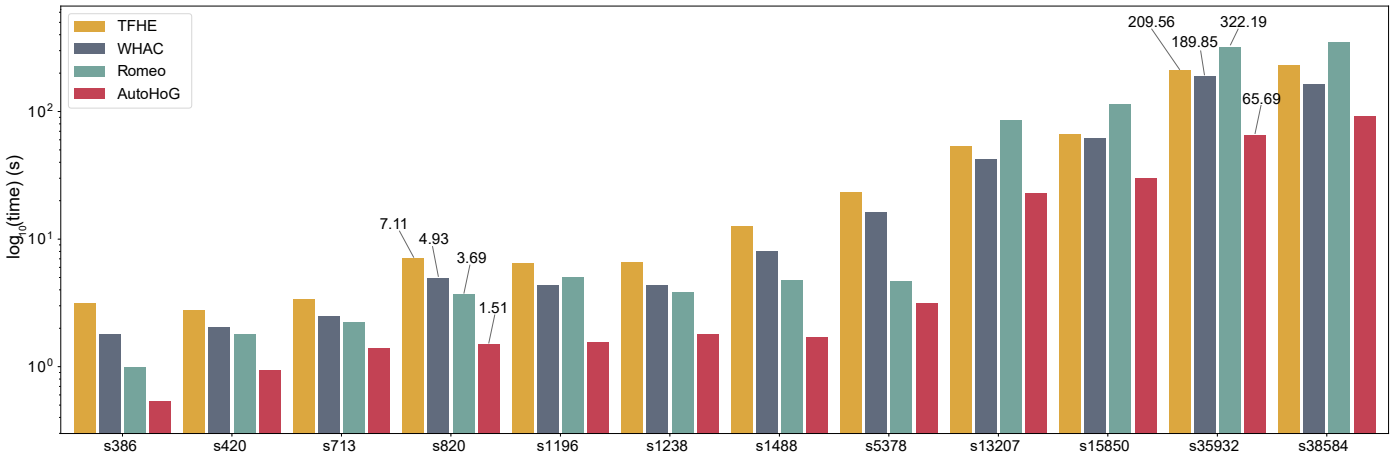

Fig. 9. Performance comparisons on ISCAS'89 benchmark circuits between AutoHoG and existing works.The execution time is the amortized execution cost per cycle over 10 cycles. Our proposed approach has demonstrated a maximum improvement of up to 5.2×.

TABLE IV
EVALUATION TIME OF ISCAS'89 BENCHMARK CIRCUITS

Benchmark	TFHE(s)	WAHC(s)	ROMEO(s)	AutoHoG(s)
s27	0.14	0.12	0.09	0.11
s298	2.06	1.48	0.20	0.60
s344	1.77	1.50	0.38	0.58
s349	1.87	1.58	0.31	0.67
s382	2.50	1.65	0.73	0.56
s386	3.16	1.81	0.99	0.54
s400	2.60	1.69	0.73	0.55
s420	2.76	2.03	1.79	0.94
s444	2.84	2.06	0.82	0.60
s510	3.50	2.76	0.74	1.02
s526	4.32	3.00	0.50	1.03
s641	2.67	1.96	2.10	1.25
s713	3.40	2.49	2.25	1.39
s820	7.12	4.94	3.69	1.50
s832	7.71	5.34	2.59	1.61
s838	5.68	4.13	2.47	1.89
s953	5.93	4.54	2.43	1.69
s1196	6.44	4.32	5.00	1.55
s1238	6.62	4.38	3.83	1.81
s1423	8.40	6.64	3.79	2.79
s1488	12.66	8.04	4.80	1.70
s5378	23.15	16.35	4.64	3.15
s9234	40.25	30.64	12.65	11.18
s13207	53.28	41.99	85.28	22.84
s15850	66.71	61.92	114.03	30.11
s35932	209.56	189.85	322.19	65.69
s38584	231.75	162.80	351.93	92.21

time. For instance, when applying AutoHoG to the c432 benchmark circuit, both algorithms are able to reduce the circuit latency from 3.00 s to 1.39 s. However, Algorithm 1 only takes 122 s, while the Z3 SMT solver takes 2223 s (roughly 18× faster). In addition, we also compare the two algorithms on the larger c3540 circuit. Here, AutoHoG completes the replacement process using Algorithm 1 in 1509 s, reducing the latency from 21.82 s to 5.60 s. In comparison, the Z3 SMT solver did not finish running within 12 hours, and we terminated its execution. Therefore, although integer programming solvers can generate better homomorphic compound gates as mentioned in Section III-B, the practical efficacy can be limited.

C. Circuit Performance Comparisons and Generation Time

We apply the generated gates to the ISCAS'85 circuit benchmark [50], and the latency results are summarized in Fig. 7. We observe that AutoHoG outperforms existing works on all circuits in ISCAS'85, with as much as 5.7× speedup compared to [4] and 4.3× over [3]. However, the concrete acceleration ratios can depend on circuit topology. For example, for the c1355 circuit, where we obtain a significant latency reduction, we discover that such reduction is mainly due to the large number of multi-stage NAND trees in the circuit. The reduction in latency is mainly due to the presence of certain structures in the circuit. This may make it difficult to fully utilize homomorphic logic gates to replace existing structures in the circuit. Despite such variations, AutoHoG is still able to outperform most existing works with an average speedup of 2.8×.

We present the time required to replace each circuit in Fig. 8. The computational cost of the gate replacement process is usually directly proportional to the size of the original circuit. Meanwhile, as indicated in Eq. (11), the impact of adding a substantial number of gates to the sequence in the Truth Table Matching process can lead to longer processing times. This issue is particularly pronounced when encountering a large number of multi-input gates within a given netlist, such as the multiple eight-input gates presented in c1908.

We also apply AutoHoG to the ISCAS'89 circuit benchmark [51], and the results are depicted in Fig. 9. For sequential circuits with flip flops, we adopt the same gate re-evaluation technique proposed in [3]. Numbers annotated

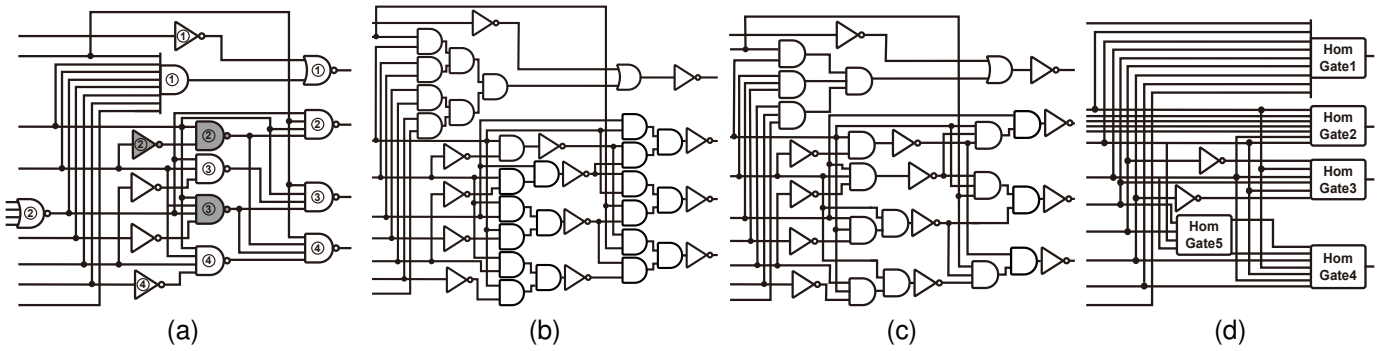

Fig. 10. The illustration of a sub-circuit in c432. (a) is the original Verilog gate-level netlist. (b) and (c) are the optimized circuits generated by Roemo and WAHC, respectively. (d) is the circuit optimized by AutoHoG based on (a). Here, gates with the same label are replaced by a single compound homomorphic gate, and gates with the same color are combined into a multi-output gate.

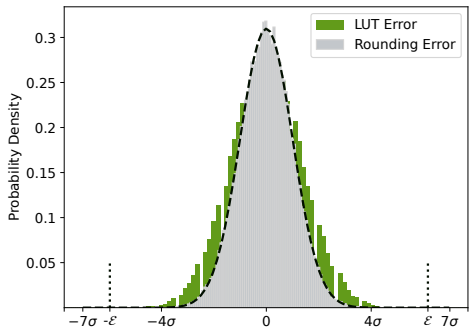

Fig. 11. The simulated probability distributions of the LUT errors and theoretical rounding errors exhibit notable similarities. The theoretical upper bound on the noise $\mathcal{E}$ lies within the range of 4 to 7 times the standard deviation (σ) of the noise, which is far enough from the simulated results.

in Fig. 9 indicate the amortized execution cost per cycle over ten clock cycles. We see that, for most reasonably-large circuits (circuits with $\geq$ 100 gates), AutoHoG outperforms existing works in ISCAS'89, achieving up to 5.2$\times$ speedup compared to [4] and 4.9$\times$ over [3]. However, we do see that for small circuits, AutoHoG exhibits a marginal slowdown compared to Romeo [3], attributed to the longer running time per gate bootstrapping due to the different parameter setups. In more complex circuits, such as s35932 and s38584, AutoHoG demonstrates a substantial advantage over [3], [29] and [4]. The full comparison results of the ISCAS'89 benchmark are summarized in Table IV.

In Fig. 10, we take a portion of the c432 benchmark, namely Module M5, as an example to better explain the comparison with prior works. Module M5 is a 9-line-to-4-line priority encoder, and Fig. 10a is its Verilog gate-level netlist. Fig. 10b illustrates the circuit converted from the Verilog code through RTL synthesis and compiled into an encrypted circuit using Romeo [3], and Fig. 10c depicts the homomorphic circuit optimized by WAHC [4]. First, we note that TFHE [29] and Romeo [3] execute nearly identical circuits as they share the same homomorphic gate library. Next, we observe that the circuit generated by [4] is slightly more optimized. In comparison, Fig. 10d illustrates the circuit optimized using AutoHoG. The gates sharing the same label in Fig. 10a are replaced by a single homomorphic gate (e.g., the gates labeled with ① are replaced by HomGate1), while gates colored the same way are combined into a multi-output gate (e.g., the gates colored in grey are combined into a two-output gate, HomGate5). Hence, we demonstrate that using AutoHoG the number of gates in c432 M5 can be reduced from 26 to 5 (more than 5$\times$ reduction in the number of gates).

D. Results on Noise Distribution

In addition to latency results, we empirically verified the proposed decryptability analyses using 100K Monte Carlo simulations for the decryption of the proposed gates, and the probability density for the error is given in Fig. 11. We note two facts. First, we confirm that the total noise in LUT evaluation is dominated by the rounding noise. Second, the theoretical upper bound on the noise $\mathcal{E}$ is far enough from the simulated results, and corresponds to a decryption failure probability that is $< 10^{-6}$.

V. Conclusion

We propose AutoHoG, an automated procedure to generate compound gates and optimize large-scale logic circuits over FHE. We formalize the mechanism of homomorphic gate design and develop gate replacement policies that help reduce the overall circuit evaluation latency. We demonstrate that, compared to the state-of-the-art homomorphic gate designs, AutoHoG can greatly boost the performance of the homomorphic evaluation over large-scale logic circuits.

References

[1] S. Bian, M. Hiromoto, and T. Sato, "Scam: Secured content addressable memory based on homomorphic encryption," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 984–989.

[2] ——, "Darl: Dynamic parameter adjustment for lwe-based secure inference," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1739–1744.

[3] C. Gouert and N. G. Tsoutsos, "Romeo: conversion and evaluation of hdl designs in the encrypted domain," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[4] K. Matsuoka, Y. Hoshizuki, T. Sato, and S. Bian, "Towards better standard cell library: Optimizing compound logic gates for tfhe," in *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2021, pp. 63–68.

[5] W.-j. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu, "Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1057–1073.

[6] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *International Conference on Machine Learning*. PMLR, 2022, pp. 12 403–12 422.

[7] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 809–826.

[8] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe," in *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*. Springer, 2021, pp. 670–699.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT*, 2017, pp. 409–437.

[10] I. Chillotti, M. Joye, D. Ligier, J.-B. Orfila, and S. Tap, "CONCRETE: Concrete operates on ciphertexts rapidly by extending tfhe," in *WAHC 2020-8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2020.

[11] R. Banno, K. Matsuoka, N. Matsumoto, S. Bian, M. Waga, and K. Suenaga, "Oblivious online monitoring for safety ltl specification via fully homomorphic encryption," in *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part I*. Springer, 2022, pp. 447–468.

[12] T. Hackenjos, F. Hahn, and F. Kerschbaum, "SAGMA: secure aggregation grouped by multiple attributes," in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 587–601. [Online]. Available: https://doi.org/10.1145/3318464.3380569

[13] K. Matsuoka, R. Banno, N. Matsumoto, T. Sato, and S. Bian, "Virtual secure platform: A five-stage pipeline processor over tfhe." in *USENIX Security Symposium*, 2021, pp. 4007–4024.

[14] A. Guimarães, E. Borin, and D. F. Aranha, "Revisiting the functional bootstrap in TFHE," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 229–253, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i2.229-253

[15] S. Gorantala, R. Springer, S. Purser-Haskell, W. Lam, R. Wilson, A. Ali, E. P. Astor, I. Zukerman, S. Ruth, C. Dibak *et al.*, "A general purpose transpiler for fully homomorphic encryption," *arXiv preprint arXiv:2106.07893*, 2021.

[16] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, p. 34, 2009.

[17] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[18] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2007, pp. 354–361.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–178.

[20] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," pp. 868–886, 2012.

[21] M. Ajtai, "Generating hard instances of lattice problems," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 99–108.

[22] D. Micciancio and O. Regev, "Worst-case to average-case reductions based on gaussian measures," *SIAM Journal on Computing*, vol. 37, no. 1, pp. 267–302, 2007.

[23] S. Bian, D. Kundi, K. Hirozawa, W. Liu, and T. Sato, "APAS: application-specific accelerators for rlwe-based homomorphic linear transformations," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 4663–4678, 2021.

[24] A. Guimarães, E. Borin, and D. F. Aranha, "MOSFHET: Optimized software for fhe over the torus," *Cryptology ePrint Archive*, 2022.

[25] S. Halevi and V. Shoup, "Design and implementation of helib: a homomorphic encryption library," *IACR Cryptol. ePrint Arch.*, 2020.

[26] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) LWE ciphertexts," in *ACNS*, 2021, pp. 460–479.

[27] L. Jiang, Q. Lou, and N. Joshi, "MATCHA: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus," *CoRR*, 2022.

[28] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *ITCS*, S. Goldwasser, Ed., 2012, pp. 309–325.

[29] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[30] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*. Springer, 2015, pp. 617–640.

[31] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *CRYPTO*, 2013, pp. 75–92.

[32] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

[33] Y. Lee, J.-W. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and H. Kang, "High-precision bootstrapping for approximate homomorphic encryption by error variance minimization," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 551–580.

[34] A. Al Badawi and Y. Polyakov, "Demystifying bootstrapping in fully homomorphic encryption," *Cryptology ePrint Archive*, 2023.

[35] Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim, "META-BTS: Bootstrapping precision beyond the limit," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 223–234.

[36] M. Albrecht, "Homomorphic encryption security standard. homomorphicencryption," org, Technical report, Tech. Rep., 2018.

[37] J. Alperin-Sheriff and C. Peikert, "Faster bootstrapping with polynomial error," in *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*. Springer, 2014, pp. 297–314.

[38] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for lut-based fpgas," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, 2006, pp. 41–49.

[39] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient fpga mapping solution," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999, pp. 29–35.

[40] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[41] J. Cong and Y.-Y. Hwang, "Simultaneous depth and area minimization in lut-based fpga mapping," in *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, 1995, pp. 68–74.

[42] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.

[43] S. Ray, A. Mishchenko, N. Een, R. Brayton, S. Jang, and C. Chen, "Mapping into lut structures," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 1579–1584.

[44] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2894–2903, 2006.

[45] A. H. Farrahi and M. Sarrafzadeh, "Complexity of the lookup-table minimization problem for fpga technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 11, pp. 1319–1332, 1994.

[46] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "Wiremap: Fpga technology mapping for improved routability and enhanced lut merging," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 2, pp. 1–24, 2009.

[47] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in lut-based fpga technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2331–2340, 2006.
[48] S. Carpov, M. Izabachène, and V. Mollimard, "New techniques for multi-value input homomorphic evaluation and applications," in *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*. Springer, 2019, pp. 106–126.
[49] H. Cramér, *Mathematical methods of statistics*. Princeton university press, 1999, vol. 43.
[50] F. Brglez, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran," in *Proc. Intl. Symp. Circuits and Systems, 1985*, 1985.
[51] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1989, pp. 1929–1934.
[52] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.
[53] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

Zhenyu Guan (Member, IEEE) received the Ph.D. degree in electronic engineering from Imperial College London, the United Kingdom, in 2013. Now, he is a professor of the School of Cyber Science and Technology at Beihang University. His current research interests include image processing and high performance computing. He has published more than 45 technical papers in international journals and conference proceedings.

Ran Mao received the B.E. degree in School of Computer Science and Engineering(School of Cybersecurity) from University of Electronic Science and Technology of China, Chengdu, China, in 2021, and is currently pursuing the Ph.D. degree at School of Cyber Science and Technology, Beihang University, Beijing, China. Her current research interests include homomorphic encryption and hardware security.

Qianyun Zhang (Member, IEEE) received her B.Sc. degree from Beijing University of Posts and Telecommunications, China, in 2014, and her Ph.D. degree from Queen Mary University of London, United Kingdom, in 2018. She is currently an Associate Professor with the School of Cyber Science and Technology, Beihang University, Beijing, China. Her research interests include wireless network security, intelligent sensing and recognition, and novel antenna designs.

Zhou Zhang received his B.S. in Beihang University, China, in 2021, and is now pursuing a M.S. degree at the School of Cyber Science and Technology, Beihang University, Beijing, China. His current research interests include encrypted database and homomorphic encryption.

Zian Zhao received the B.E. degree in electronic engineering from Beihang University in 2020, where he is currently pursuing the Ph.D. degree with the School of Cyber Science and Technology. His current research interests include applied cryptography and homomorphic encryption compilers.

Song Bian (Member, IEEE) Song Bian is currently an associate professor at Beihang University. His main areas of interest include fully homomorphic encryption, privacy-preserving computing and domain-specific hardware accelerator. He received B.S. from University of Wisconsin-Madison in 2014. He received M.S. and Ph.D. from Kyoto University, in 2017 and 2019, respectively. He was an assistant professor at Kyoto University from 2019 to 2021. He served as technical committee members/reviewers for top-tier international conferences/journals across different fields of studies, including CVPR, IEEE TIFS and IEEE TCAD. He is a member of ACM and IEEE.