SALUS: Large-Scale Homomorphic Circuit Synthesis via Logic-Aware LUT Optimization

Ran Mao, Zhou Zhang, Zian Zhao, Zhenyu Guan, *Member, IEEE*, Peng Yin✉, Song Bian✉, *Member, IEEE*

***Abstract*—Recently, growing interests are developed in optimizing fully homomorphic encryption (FHE) circuits to enable Boolean function evaluations over ciphertexts. While existing works utilize functional bootstrapping (FBS) to efficiently evaluate logic gates, the evaluation efficiency for large-scale circuits remains limited. Recent advances introduce a fast ciphertext conversion method, making it feasible to evaluate look-up tables (LUTs) over homomorphic multiplexer operation. In this work, we propose a new circuit synthesis framework, SALUS, which automatically generates and evaluates gate-level graphs over homomorphic LUTs given an input Boolean circuit. We apply the binary decision diagram (BDD) reordering method and multi-value refresh techniques to efficiently evaluate complex LUTs. Additionally, we propose a heuristic algorithm to merge LUTs in a given circuit into multi-output LUTs. In the experiments, we examine the efficiency of SALUS using a wide range of benchmark suites, including the EPFL and ISCAS benchmark circuits. We show that SALUS achieves a maximum reduction of up to $26\times$ in computational latency compared to state-of-the-art homomorphic circuit synthesis method. Furthermore, we evaluate real-world applications, e.g., image filtering and matrix multiplication, and achieve an average speedup of $8.6\times$ (with a maximum speedup of $24\times$) compared to the FBS-based method.**

***Index Terms*—fully homomorphic encryption, circuit synthesis, privacy-preserving computation.**

I. Introduction

In recent years, with the increasing demand for privacy-preserving solutions, fully homomorphic encryption (FHE) becomes a critical technique for secure data computation. FHE enables computations on encrypted data without decryption, ensuring privacy protection throughout the computation process. Homomorphic encryption schemes can be broadly categorized into word-based schemes and Boolean-based schemes. Word-based schemes, such as BFV [1], BGV [2], and CKKS [3], can leverage single-instruction-multi-data (SIMD) capabilities [4]. In contrast, Boolean-based schemes, such as FHEW [5] and TFHE [6], reduce the encryption parameter to enable fast bootstrapping. Boolean-based schemes allow the evaluation of arbitrary functions by transforming the high-level code into homomorphic circuits. Due to the efficiency of bootstrapping, Boolean-based schemes are widely used in privacy-preserving applications, including homomorphic processors [7], [8], transciphering [9]–[11], and encrypted database queries [12], [13].

Corresponding author is Song Bian and Peng Yin.

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

P. Yin is with Woodpecker Innovation Center, Beijing 100048, China (e-mail: pedro_yin@163.com).

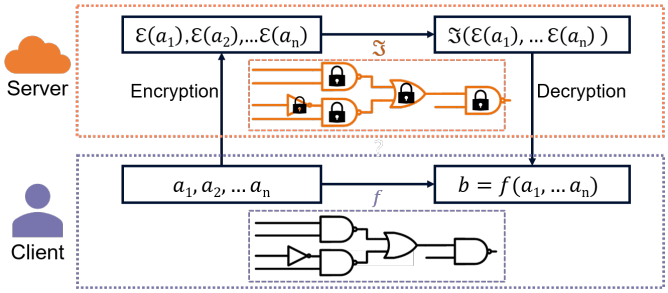

Fig. 1. The overview of a round of homomorphic Boolean circuit evaluation.

To evaluate circuits over FHE, Boolean-based schemes leverage functional bootstrapping (FBS) to evaluate logic gates such as NAND [6]. As illustrated in Figure 1, the client encrypts the input data and sends them to the server, which evaluates a public circuit by performing homomorphic logic gate evaluations. The resulting ciphertext is then returned to the client, who obtains the result of the circuit computation upon decryption. The method behind homomorphic circuit evaluation is similar to computing over traditional electronic circuit, where complex Boolean functions are realized by evaluating interconnected logic gates.

To optimize Boolean-based FHE schemes and improve the efficiency of homomorphic circuit evaluation, many existing works [14]–[16] adapt design philosophies from integrated circuits to homomorphic evaluation. Moreover, methods such as [17]–[21] focus on optimizing homomorphic gate evaluation, aiming to evaluate compound logic gates using fewer FBS operations. Meanwhile, since cell libraries with homomorphic gates differ from conventional cell libraries, traditional circuit design methodologies are not fully applicable to homomorphic circuits. Hence, various approaches [22]–[24] targeting homomorphic circuit synthesis apply circuit mapping techniques specifically designed for homomorphic gates. However, we observe that all existing homomorphic circuit evaluation methods rely on FBS for gate evaluation, which leads to limitations in the types of gates that can be evaluated due to parameter constraints. This restricts evaluation performance, particularly for complex Boolean tasks.

In this work, we propose SALUS, a new framework for evaluating Boolean functions through homomorphic LUT evaluation. To overcome the parameter constraints of FBS-based methods, SALUS adopts a homomorphic multiplexer (`CMUX`) tree in the leveled homomorphic encryption (LHE) mode [25] to evaluate LUTs with a larger number of inputs. Since the ciphertext formats of LHE inputs and outputs differ, we apply

a fast ciphertext conversion method [26], [27] to make it feasible to utilize CMUX-based LUTs. We enhance existing homomorphic LUT construction through BDD reordering and multi-value refresh techniques. Furthermore, we introduce a logic similarity based LUT merging heuristic to further improve performance. As a result, SALUS substantially enhances computational efficiency compared to approaches that rely solely on FBS-based homomorphic gates. The insights and contributions of this work are summarized as follows:

- **An Encrypted Logic Synthesis Framework:** We propose a novel logic synthesis framework for circuit mapping to homomorphic LUTs, utilizing a combination of FBS and CMUX techniques. To ensure the correctness of the circuit evaluation, we study the noise characteristics of SALUS and provide a parameter setup based on the requirements of the client.
- **Homomorphic LUT Optimization:** We develop an improved method for evaluating single-output and multi-output LUTs, which enables efficient evaluation under a predefined set of FHE parameters. We apply the BDD reordering method to minimize the size of the CMUX tree and employ the multi-value refresh technique to reduce the number of noise refresh operations during LUT evaluation.
- **LUT Merging Heuristics:** We propose a heuristic algorithm that selectively merges nodes into multi-output LUTs, optimizing the given circuit. The algorithm uses a similarity matrix to prioritize merging LUTs with similar logic. It guarantees that the logical dependencies of the circuit are preserved, while reducing the overall computation overhead.
- **Application to Boolean Tasks:** In the experiments, we demonstrate that our improved LUT outperforms existing methods in evaluating complex circuits. For the microbenchmarks with single LUTs, our method achieves up to a $32\times$ and $22\times$ speedup compared to the FBS-based method and the previous CMUX-based method, respectively. For the large benchmark circuits, our method reduces the evaluation latency by up to $26\times$ when compared to state-of-the-art solutions[1].

The rest of this paper is organized as follows. First, in Section II, we introduce the preliminaries in the TFHE scheme and homomorphic operators. Second, in Section III, we provide an overview of the SALUS framework, including security analysis. Third, in Section IV, we present our LUT evaluation optimization and a heuristic approach for merging LUTs. Fourth, in Section V, we provide a brief analysis of the noise growth in SALUS and discuss the parameter setup. Fifth, the performance of SALUS is illustrated through microbenchmark experiments and large benchmark circuits in Section VI. Finally, we conclude our work in Section VII.

II. Preliminaries

In this section, we introduce the basic notations and provide an overview of the ciphertext types and the key homomorphic operators. Following that, we will present related work on logic synthesis.

A. Notations

We use lowercase letters with a tilde to represent polynomials (e.g., $\widetilde{a}$), boldface lowercase letters for vectors (e.g., $\boldsymbol{a}$), and capital letters for matrices (e.g., A). The integer p denotes the plaintext modulus, and the integer q/Q represents the ciphertext modulus. The integer n refers to the lattice dimension of integer LWE ciphertexts, while N represents the lattice dimension of RLWE ciphertexts. We denote the set of integers modulo q by $\mathbb{Z}_q$. R_N and $R_{N,Q}$ denote the quotient rings $\mathbb{Z}[X]/\left(X^N+1\right)$ and $\mathbb{Z}[X]/\left(X^N+1\right) \bmod Q$, respectively. For a comprehensive list of notations and terminologies, please refer to Section A.

B. The TFHE Scheme

The TFHE scheme employs three distinct ciphertext types: learning-with-errors (LWE) ciphertext LWE, ring-learning-with-errors (RLWE) ciphertext RLWE and ring-Gentry-Sahai-Waters (GSW) ciphertext RGSW. In addition, there is a specific ciphertext type in the LHE mode, referred to as the ring-level (RLev) ciphertext RLev, as introduced in [28]. These ciphertext types are outlined as follows:

- $\mathsf{LWE}_{\mathbf{s}}^{n,q}(m)$: The LWE ciphertext encrypts a single message $m \in \mathbb{Z}_p$ using a secret key $\mathbf{s} \in \mathbb{Z}_2^n$, which is a binary vector randomly chosen from a uniform distribution. It is defined as follows:
$$\mathsf{LWE}_{\mathbf{s}}^{n,q}(m) = (\mathbf{b}, \mathbf{a}) = (\langle \mathbf{a}, \mathbf{s} \rangle + \Delta m + e, \mathbf{a}) . \quad (1)$$
where $\mathbf{a} \in \mathbb{Z}_q^n$ is a uniformly random vector, and e represents fresh noise sampled from the error distribution χ_σ with standard deviation σ. The term Δ is a scaling factor used to separate the encoded message from the noise.
- $\mathsf{RLWE}_{\widetilde{s}}^{N,Q}(\widetilde{m})$: The RLWE ciphertext encrypts a polynomial of encoded messages $\widetilde{m} \in R_{N,P}$ under a secret key $\widetilde{s} \in R_{N,Q}$, which can be formulated as:
$$\mathsf{RLWE}_{\widetilde{s}}^{N,Q}(\widetilde{m}) = (\widetilde{b}, \widetilde{a}) = (-\widetilde{a} \cdot \widetilde{s} + \Delta \widetilde{m} + \widetilde{e}, \widetilde{a}) . \quad (2)$$
- $\mathsf{RLev}_{\widetilde{s}}^{N,Q,\mathsf{Bg},d}(m)$: The RLev ciphertext is a tuple of RLWE ciphertexts that encrypts a decomposed message $m \in \mathbb{Z}_p$, characterized by a decomposition base Bg and a decomposition length d. It can be expressed as:
$$\mathsf{RLev}_{\widetilde{s}}^{N,Q,\mathsf{Bg},d}(m) = \left(\mathsf{RLWE}_{\widetilde{s}}^{N,Q} \left(\left\lceil \frac{Q}{\mathsf{Bg}^j} \right\rfloor \cdot m \right) \right)_{j \in d} . \quad (3)$$
- $\mathsf{RGSW}_{\widetilde{s}}^{N,Q,\mathsf{Bg},d}(m)$: The RGSW ciphertext encrypts a message $m \in \mathbb{Z}_p$ under a secret key $\widetilde{s} \in R_{N,Q}$, which is defined as:
$$\mathsf{RGSW}_{\widetilde{s}}^{N,Q,\mathsf{Bg},d}(m) = (\widetilde{B}, \widetilde{A}) = \left(\widetilde{A} + m \cdot H, \widetilde{A} \right) , \quad (4)$$
where $H = I \otimes D_{\mathsf{Bg},d}$, with $D_{\mathsf{Bg},d} = \left[\frac{1}{\mathsf{Bg}} \quad \cdots \quad \frac{1}{\mathsf{Bg}^d} \right]^T$ and I denoting the identity matrix. The matrix H is referred to as the gadget matrix. An RGSW ciphertext can be viewed as a composition of $2d$ RLWE ciphertexts.

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

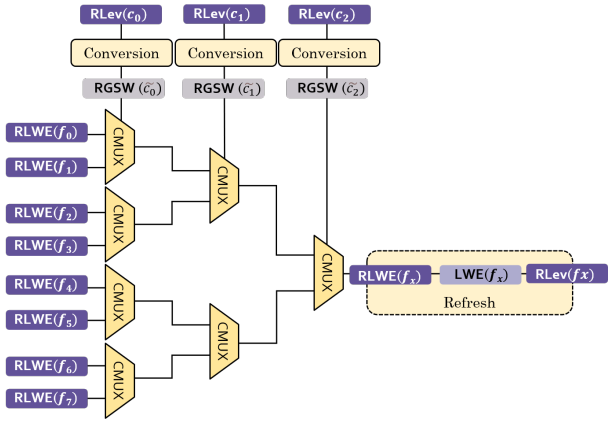

Fig. 2. The method of CMUX-based homomorphic LUT evaluation.

C. Homomorphic Operators

Here, we introduce the fundamental operators used throughout this work, along with two Boolean function evaluation operators. For clarity, when specific parameters are not essential to discuss, we abbreviate the ciphertext notations as $\mathsf{LWE}(m)$, $\mathsf{RLWE}(\widetilde{m})$,$\mathsf{RLev}(m)$ and $\mathsf{RGSW}(\widetilde{m})$.

Fundamental Operators:

- External Product: This operation is a special homomorphic multiplication, with the output satisfied as follows:

$$\mathsf{RLWE}(\widetilde{m}_0) \times \mathsf{RGSW}(\widetilde{m}_1) = \mathsf{RLWE}(\widetilde{m}_0 \cdot \widetilde{m_1}). \quad (5)$$

- CMUX: CMUX operator is a homomorphic multiplexer. Given a selection signal $\mathsf{RGSW}(\widetilde{C})$ where $\widetilde{C} \in \{0,1\}$ and two inputs $\mathsf{RLWE}(\widetilde{d_0})$ and $\mathsf{RLWE}(\widetilde{d_1})$, we have:

$$\begin{aligned}\mathsf{RLWE}(\widetilde{d_C}) &= \mathtt{CMUX}(\mathsf{RGSW}(\widetilde{C}), \mathsf{RLWE}(\widetilde{d_0}), \mathsf{RLWE}(\widetilde{d_1})) \\ &= \begin{cases} \mathsf{RLWE}(\widetilde{d_0}) & \text{if } \widetilde{C} = 0, \\ \mathsf{RLWE}(\widetilde{d_1}) & \text{if } \widetilde{C} = 1. \end{cases}\end{aligned} \quad (6)$$

- Key Switching: In this paper, key switching specifically refers to public key switching where the public function is the identity function. This operator takes $\mathsf{LWE}^{N,Q}(m)$ as input and converts it into $\mathsf{LWE}^{n,q}(m)$.
- Blind Rotation: Blind rotation performs a negacyclic rotation on a given polynomial test vector ($\mathsf{TV}(x)$) by m positions. Here, $\mathsf{TV}(x) = \sum_{i=0}^{n-1} \tau_i x^i$ is a plaintext polynomial, while m is encrypted as $\mathsf{LWE}(m)$. Multiplying $\mathsf{TV}(x)$ with $\mathsf{LWE}(m)$ gives us:

$$\mathsf{TV}(x) \cdot x^{\mathsf{LWE}(m)} = \mathsf{RLWE}(\sum_{i=0}^{n-1} \tau_i x^i \cdot x^m \bmod x^n + 1) \quad (7)$$

- Sample Extraction: This operation involves extracting a specific coefficient from an RLWE ciphertext at a given index i. This operation does not introduce additional noise and can be performed with negligible computational cost.
- Conversion: The ciphertext conversion from $\mathsf{RLev}(m)$ to $\mathsf{RGSW}(m)$, which is a part of circuit bootstrapping proposed in [27].

Boolean Operators:

- HomGate: The evaluation of a homomorphic logic gate consists of four main steps: (i) Linear Combination: Computing a weighted sum of the input LWE ciphertexts, resulting in one LWE that encrypts an integer. (ii) Key Switching: Reducing the dimension of LWE from N to n. (iii) Blind Rotation: Rotating the test vector $\mathsf{TV}(x)$, which is generated from the table TableT. The construction of TableT is derived from the truth table of the gate, as proposed in [18]. (iv) Sample Extraction: Extracting the 0-th coefficient from the rotated RLWE to produce the final result. In this process, blind rotation refreshes the noise of the ciphertext, thereby enabling the evaluation of circuits with arbitrary depth. Blind rotation also remains the most time-consuming part of the gate evaluation.
- HomLUT: The process of homomorphic LUT evaluation takes multiple RLev ciphertexts as input. As shown in Figure 2, HomLUT involves three main steps: i) Conversion: Converting the input RLev ciphertexts into RGSW ciphertexts. ii) CMUX tree: Performing LUT evaluation through a CMUX tree. iii) Refresh: Refreshing the noise of the output ciphertext and converting it back to RLev ciphertexts. In ii), the selectors of the CMUX tree are the input ciphertexts obtained after Conversion, while the data inputs consist of a series of RLWE ciphertexts, each encrypting a value from the LUT. Step iii) involves two specific functionalities: converting the ciphertext back to the RLev ciphertext and refreshing the noise in the ciphertext. In this step, the output RLWE ciphertext is first transformed into an LWE ciphertext through sample extraction. Subsequently, a blind rotation operation is applied to decompose the ciphertext into the RLev format. Here, PBSmanyLUT [28] is implemented, allowing a single blind rotation to produce d decomposed RLWE ciphertexts.

D. Related Works

*1) **Logic synthesis**:* Logic synthesis is the process of converting high-level hardware description language (HDL) into gate-level representations, aiming to improve design metrics such as area and delay.

Technology mapping is an important step in the synthesis process of field-programmable gate array (FPGA) design. For FPGA based on look-up tables, there are several mapping method based on cut enumeration [29]. For FPGA with K-input LUTs, mapping involves computing K-feasible cuts for each two-input internal node. Some structural algorithms, such as FlowMap [30] and CutMap [31], use the maximum-flow algorithm to find one optimal cut per node. Some hardware synthesis tools like ABC [32] employ various optimization techniques, including priority cut mapping [33] and cut pruning [29], [34], to further reduce memory usage. Building on these techniques, recent works have explored multiple strategies to minimize the circuit. For example, [35], [36] utilizes Quantified Boolean Formulas (QBF) and Boolean satisfiability problem (SAT) solvers for resynthesis. [37] applies circuit simulation for substitution. [38], [39] incorporates machine

learning (ML) algorithms to automatically search different optimizers for various objectives.

Some modern FPGA architectures adopt multi-output LUTs. An expanded concept called *KL*-cuts has been introduced [40], which represents cuts with no more than K inputs and L outputs. Recent works, such as [41], explore the use of dual-output LUTs during technology mapping to achieve higher merging rates. Additionally, [42] proposes technology mapping algorithms that support multi-output cells, offering new opportunities for optimization. However, the adoption of multi-output LUT during technology mapping introduces significant complexity to both cut matching and cut computation, making it challenging to scale for larger K and L. Consequently, most *KL*-cuts mapping methods are limited to relatively small K and L, such as $K = 6$ and $L = 3$. Some existing tools adopt a later packing phase to merge LUTs. Although the merge rate may not be as high as that of the heuristic methods, this approach is effective for merging large LUTs.

BDD minimization [43] serves as another effective logic optimization technique. An arbitrary Boolean function can be represented as a BDD, where each node corresponds to a 2-to-1 multiplexer, and the inputs serve as the selectors of the multiplexers. However, unoptimized BDDs contain a significant number of redundant nodes, leading to inefficiencies. The order of inputs greatly influences the effectiveness of BDD reduction. Notably, BDD reordering is an NP-complete problem [44]. As a solution, various reordering technologies have been proposed, including the exact ordering methods and heuristic algorithms such as dynamic sifting reordering and window permutation [45]. Despite these advancements, the computational complexity of reordering increases significantly with the increasing number of inputs, making it infeasible for large-scale designs. Beyond computational complexity, directly applying BDD minimization to homomorphic circuit synthesis introduces additional challenges. In homomorphic evaluation, every multiplexer node in the BDD corresponds to a `CMUX` operation that accumulates ciphertext noise. When the entire circuit is represented as a single BDD, noise continuously grows along the evaluation path until it exceeds the decryption threshold. Although `Refresh` operations can bring the noise back within the acceptable range, determining appropriate refresh boundaries within a large BDD remains difficult. For these reasons, we apply BDD reordering only as a local optimization for individual LUTs, whose sizes are naturally bounded by parameter constraints, rather than as a global circuit representation.

2) ***Homomorphic circuit synthesis****:* Previous homomorphic circuit evaluation primarily relies on functional bootstrapping to evaluate LUTs. As a result, circuit performance is directly determined by the total number of FBS. Consequently, optimizing FHE circuits is similar to area optimization in traditional logic synthesis. Previous works, such as [15] and [16], utilize existing logic synthesis tools such as Yosys [46], to convert Boolean circuits into LUT networks. Building upon these efforts, subsequent works introduce various improvements to enhance the efficiency of FHE circuit. For example, [20] utilizes multiple fan-in gates with reduced bootstrapping to decrease circuit depth. [21] defines a new set of bits encoding method, enabling the evaluation of more complex Boolean functions using a single FBS. [18] proposes a general framework for automatically generating compound logic gates under given parameters, replacing a sequence of gates to reduce the number of FBS in the circuit. [19] expands the plaintext space of encrypted data to further optimize encryption parameters. Moreover, [22] introduces a LUT grouping method for evaluating multi-output logic gates based on the multi-value bootstrapping idea [47]. [23] establishes a homomorphic gate set for area-oriented technology mapping and proposes a novel technology mapping algorithm to maximize the utilization of multi-output gates. Recent work [24] extends FBS-based LUTs to word-wise operations, which leverages three evaluation modes: Boolean gate, small-precision LUT and high-precision LUT. Despite the progress, FBS-based methods remain constrained by parameter limitations, restricting the feasible LUT size. Challenges persist in the performance of Boolean circuit evaluation, especially for large-scale circuits.

III. SALUS FRAMEWORK

Here, we first provide an overview of SALUS workflow in Section III-A, where we introduce each component in our framework. Next, we establish a threat model for the framework in Section III-B.

A. Procedure Overview

An overview of the SALUS procedure is illustrated in Figure 3, where the workflow is divided into three main phases: the generation phase, the synthesis phase, and the evaluation phase. In SALUS, the client encrypts the data and sends it to the server along with the HDL code and a set of predefined parameter requirements. The generation phase first determines the appropriate parameter settings. A set of homomorphic Boolean operators that meet the specified parameter constraints is then generated. Next, the synthesis phase converts the HDL code into a homomorphic circuit based on the generated Boolean operators. Finally, the evaluation phase performs the homomorphic circuit evaluation under the given input ciphertext. The computation results are returned to the client, who decrypts them to obtain the final outputs. In what follows, we provide further details on each phase.

1) ***Generation phase****:* The generation phase consists of a parameter setup block and an operator generation block. The parameter setup block includes both noise analysis and security analysis. A set of encryption parameters is given, including N, Q, n, q and decomposition parameters Bg and d. The analysis determines the maximum number of inputs $\mathcal{M}_{in}$ and outputs $\mathcal{M}_{out}$ that a single `HomLUT` can support, as well as $\ell_{\max}$, the largest `TableT` size that a single `HomGate` can evaluate, all while ensuring compliance with the required decryption accuracy and security standards from client. It is worth noting that larger encryption parameters enable the evaluation of more complex Boolean functions, but at the cost of increasing evaluation time for a single `HomGate` or `HomLUT`. Upon receiving $\mathcal{M}_{in}$, $\mathcal{M}_{out}$ and $\ell_{\max}$, the operator generation block generates gates and LUTs according to the

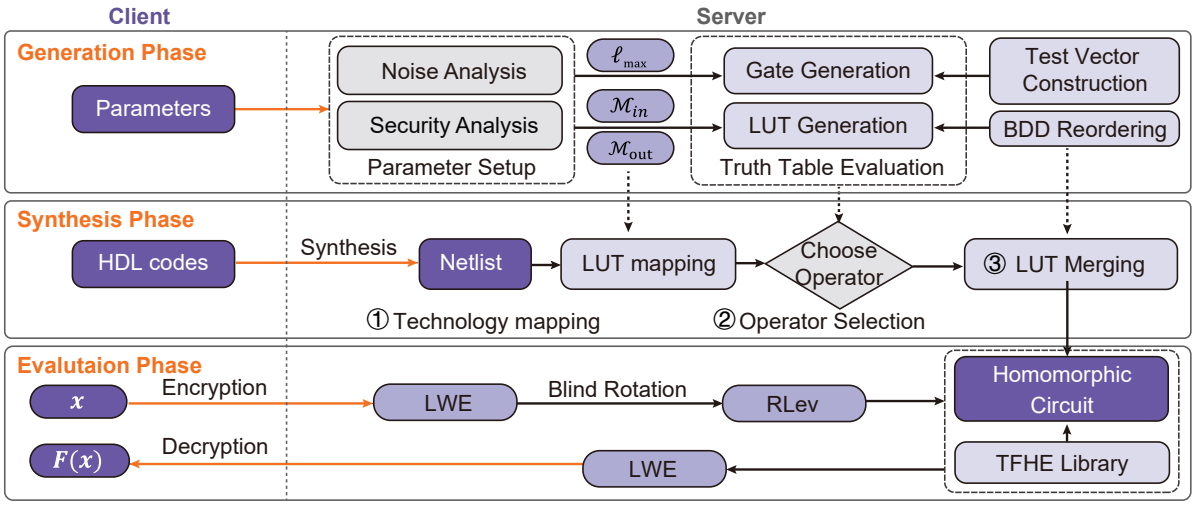

Fig. 3. A conceptual illustration of the SALUS framework.

parameter constraints. For homomorphic gates, `TableT` is constructed such that $|\mathtt{T}| \leq \ell_{\max}$ is satisfied. For homomorphic LUTs, all LUTs that meet the input constraint $\mathcal{M}_{in}$ and the output constraint $\mathcal{M}_{out}$ are directly generated. The `CMUX` tree reduction is then applied to reduce the number of `CMUX` operations. More details are discussed in Section IV-A.

*2) **Synthesis phase**:* The synthesis phase can be divided into three steps: ① Technology Mapping, ② Operator Selection, and ③ LUT Merging. We provide a detailed introduction to each step below.

① **The Technology Mapping Step**. Upon receiving the data, the HDL code is first synthesized into a Boolean circuit using Yosys [46]. Then, we apply technology mapping in ABC [32] to map the Boolean gates into LUTs. The LUT size is constrained by the parameter K, which is set to $\mathcal{M}_{in}$, determined during the generation phase. We iteratively invoke ABC with different synthesis and mapping strategies. This iterative process continues until no further reduction in the number of LUTs in the circuit is observed. To verify the correctness of mapping, we use the `cec` command in ABC to perform combinational equivalence checking (CEC) [48]. The CEC procedure constructs a miter circuit from the original and mapped networks and employs SAT solving to prove their equivalence.

② **The Operator Selection Step**. In this step, we traverse each LUT in the mapped circuit and choose an appropriate Boolean operator for evaluation. Specifically, we first check whether the LUT can be represented as a `HomGate`, generated during offline. If so, the node is marked for evaluation using `HomGate`. Otherwise, the node is evaluated using `HomLUT`. Remarkably, the input ciphertext types of `HomGate` and `HomLUT` are LWE and RLev, respectively. If two `HomLUTs` or two `HomGates` are connected in a homomorphic circuit, they can be directly linked. If a `HomGate` is followed by a `HomLUT`, the `TableT` of `HomGate` should be modified to simultaneously evaluate the truth table and the ciphertext decomposition. Conversely, when a `HomLUT` is followed by a `HomGate`, the `Refresh` step of the `HomLUT` can be omitted. Instead, the computation result of the `CMUX` tree can be directly used as the input of the `HomGate`.

③ **The LUT Merging Step**. To better leverage the ability of `HomLUT` to compute multiple outputs simultaneously, we introduce a LUT-merging method in the workflow. This step applies a heuristic to merge specific `HomLUT` into multi-output `HomLUT`, further reducing evaluation latency. The correctness of LUT merging follows from the fact that it is a purely structural transformation. Merging several LUT nodes does not modify the Boolean functions they represent, and the operation simply bundles multiple outputs into a single node with a unified input set. Moreover, BDD reordering in `HomLUT` only affects the internal CMUX topology but never changes the function semantics.

The resulting homomorphic circuit is a LUT network represented as a directed acyclic graph (DAG), where each node corresponds to either a `HomGate` or a `HomLUT`, and the edges capture the dependencies among nodes.

*3) **Evaluation phase**:* To reduce communication overhead, the input data are encrypted as LWE ciphertexts and transmitted from the client to the server. Since `HomLUT` requires RLev ciphertexts as inputs, the received LWE ciphertexts are first converted to RLev ciphertexts using blind rotation, which evaluates the decomposition of the LWE ciphertexts. After receiving the input ciphertext and the encrypted circuit from the synthesis phase, each gate or LUT is evaluated using the TFHE library. The computation results are then returned to the client for decryption.

B. Threat Model and Security Guarantee

The threat model for SALUS is similar to that of most existing FHE outsourcing frameworks [16], [18], [24], [49], [50], where the primary security objective is to protect the outsourced computation data owned by the client against the server. The server is modeled as a semi-honest adversary, which does not actively attack the client or manipulate the computation process but passively observes the ciphertexts in an attempt to infer the private data.

Since the framework operates entirely on TFHE ciphertexts without introducing additional security vulnerabilities, the security of SALUS is ensured by the TFHE scheme. TFHE achieves IND-CPA security [51], ensuring the ciphertexts remain indistinguishable under chosen-plaintext attacks. As it is well established that IND-CPA security implies resilience against semi-honest adversaries, SALUS provides strong security guarantees for outsourced data.

Note that the level of security is primarily influenced by the selection of encryption parameters, including the modulus, ciphertext dimension, and noise level in the ciphertexts. Generally, larger parameter sizes and lower noise levels offer stronger security assurances but at the cost of increased computational complexity.

C. Cost Model

We represent the synthesized circuit as a DAG $G = (V, E)$, where each node $v \in V$ is annotated with two attributes: a type label $\tau(v) \in \{\texttt{HomGate}, \texttt{HomLUT}\}$ specifying the operator type, and a Boolean function f_v represented as a truth table. The edge set E encodes the data dependencies among nodes, where $(u, v) \in E$ indicates that the output of node u serves as an input to node v. Based on the topological order of G, the nodes can be partitioned into L levels $\{V_1, V_2, \ldots, V_L\}$, where nodes within the same level can be executed in parallel. P denotes the maximum parallelism supported by the target platform. The total evaluation cost of a homomorphic circuit is given by

$$T_{\text{circuit}} = \sum_{l=1}^{L} \left\lceil \frac{|V_l|}{P} \right\rceil \cdot \max_{v \in V_l} T(\tau(v), f_v), \tag{8}$$

where $|V_l|$ is the number of nodes at level l, and $T(\tau(v), f_v)$ denotes the evaluation time of node v. The evaluation time $T(\tau(v), f_v)$ depends on both the operator type $\tau(v)$ and the Boolean function f_v. For nodes with $\tau(v) = \texttt{HomLUT}$, the evaluation cost varies with the number of inputs k and the number of CMUX nodes required, both of which are determined by the structural complexity of f_v. More details are provided in Section IV. For nodes with $\tau(v) = \texttt{HomGate}$, the evaluation time is dominated by a single FBS operation. The FBS cost is determined by the encryption parameters rather than the specific Boolean function being evaluated [18].

This cost model differs from traditional FPGA synthesis. In FPGA architectures, the evaluation time of each LUT is essentially constant, so optimization efforts mainly focus on reducing the overall LUT count to minimize area or decreasing the critical-path depth to improve delay. In homomorphic circuits, however, the evaluation time varies across different nodes. Therefore, the synthesis workflow in SALUS aims to optimize the circuit size and depth, as well as the per-node evaluation cost $T(\tau(v), f_v)$.

IV. LUT Optimization

A. Single-output LUT Optimization

Here, we introduce our optimization for single-output LUT evaluation. As mentioned in Section II-C, the evaluation of HomLUT is divided into three steps: Conversion, CMUX tree evaluation and Refresh. For a k-input LUT, the CMUX tree consists of $2^k - 1$ CMUX nodes, thus the total evaluation time can be expressed as:

$$T_{\text{S-HomLUT}} = k \cdot T_{\text{conv}} + (2^k - 1) \cdot T_{\texttt{CMUX}} + T_{\texttt{Refresh}}. \tag{9}$$

We observe that although $T_{\texttt{CMUX}}$ remains relatively short, the total evaluation time grows exponentially with k, leading to a significant increase in evaluation time for large-fan-in LUTs.

To overcome the exponential latency growth, we apply the BDD reordering method to reduce the number of CMUX operations. The CMUX tree and BDD share structural similarities. Both represent a Boolean function as a binary tree, in which each internal node selects between two subtrees according to an input variable, while the leaf nodes store the corresponding output values. This structural equivalence enables the application of BDD reordering techniques to reduce the number of CMUX operations within each HomLUT. Since BDD reordering only changes the variable ordering without increasing the tree depth, it does not introduce additional noise accumulation and therefore does not compromise decryption correctness. For clarity, we use the term BDD to refer to the CMUX tree within HomLUT in the following discussion. BDD exhibits significant structural redundancy, including identical subtrees and don't-care nodes. Removing these redundant

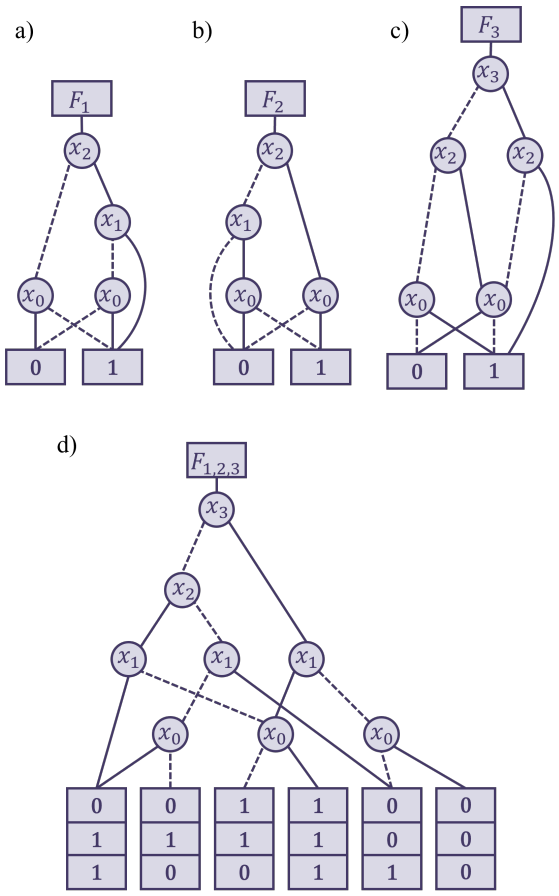

Fig. 4. An example of three BDDs merged into a single MTBDD, achieving the node count reduction from 13 to 8.

nodes does not affect the evaluation result while reducing the computational overhead. Specifically, a reduced BDD follows two rules:

① No two distinct nodes represent the same variable with identical left and right successors.
② No node has identical left and right successors.

The effectiveness of CMUX tree reduction is highly dependent on the variable ordering, as different input orders determine the structure of the tree and, consequently, the number of removable redundant nodes. Here, we employ dynamic sifting in CUDD [52] to determine the variable ordering. Although computing an optimal BDD variable ordering is NP-complete in the general case [44], dynamic sifting provides an efficient and widely used heuristic. The algorithm iteratively moves each variable across the ordering to reduce the BDD size, with a time complexity of $O(n \cdot |G|)$, where n is the number of variables and $|G|$ denotes the BDD size. While dynamic sifting does not guarantee a globally optimal ordering, prior empirical studies have shown that it typically achieves results within a small constant factor of the optimum for practical Boolean functions [45]. In SALUS, the bounded input size of each LUT (typically < 16) keeps the search space tractable. Moreover, Boolean functions generated by logic synthesis frequently exhibit structural regularities such as symmetric inputs or decomposable subfunctions, which further enhance the effectiveness of BDD reduction under variable reordering.

If a k-input LUT is reduced to m nodes after CMUX tree reduction, the evaluation time becomes:

$$T_{\texttt{S-HomLUT}} = k \cdot T_{\text{conv}} + m \cdot T_{\texttt{CMUX}} + T_{\texttt{Refresh}}. \quad (10)$$

Here, m is determined by the specific logic of the Boolean function. For functions with high symmetry, the CMUX tree may contain more redundant nodes, allowing for significant reduction.

B. Multi-output LUT Optimization

In this section, we propose optimizations for multi-output LUT evaluation and then introduce a heuristic for merging single-output LUTs into multi-output LUTs.

*1) **Evaluation optimization for multi-output LUTs**:* The RLWE encryption scheme allows the encoding of N values simultaneously. By packing different LUT values within RLWE, we can evaluate multiple functions through one CMUX tree. For LUT with k inputs and l outputs, the evaluation time can be expressed as:

$$T_{\texttt{M-HomLUT}} = k \cdot T_{\text{conv}} + (2^k - 1) \cdot T_{\texttt{CMUX}} + l \cdot T_{\texttt{Refresh}}. \quad (11)$$

Similar to the single-output LUT, the CMUX tree size can also be reduced using BDD reordering. Moreover, in multi-output LUTs, the Refresh step is also time-consuming. Hence, we apply the multi-terminal CMUX tree and multi-value refresh methods to reduce the computational overhead.

Multi-terminal CMUX Tree: Multi-terminal BDDs (MTBDDs) extend BDDs to represent multi-output logic functions, where terminal nodes can take a range of integer values instead of being limited to 0 and 1. We observe that multiple ciphertexts can be packed into the RLWE ciphertexts, enabling

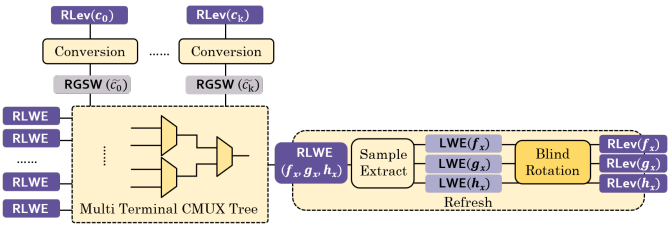

Fig. 5. The method of multi-output LUT evaluation.

a multi-terminal CMUX tree. Merging multiple BDDs into a single MTBDD can sometimes reduce the overall BDD size, particularly when the BDDs share a large number of common nodes. During the merging process, the intersection of all function inputs is taken as the new input set, and multiple new truth tables are computed accordingly.

Figure 4 illustrates an example of merging multiple Boolean functions into a single multi-output function. The three functions F_1, F_2 and F_3 are represented as separate BDDs with a consistent input ordering, as shown in Figure 4(a), (b), and (c). Specifically, F_1 and F_2 share the same inputs x_0, x_1, x_2, while F_3 only shares x_0, x_2, along with an additional input x_3. We take the union of all the inputs x_0, x_1, x_2, and x_3 as the inputs for the new MTBDD, and then recompute the truth tables of the three functions under the new input set. The new truth tables are packed as $\{0(000), 1(001), 2(010), 3(011), 6(110), 7(111)\}$, serving as the terminals of the MTBDD. Then, the MTBDD is reordered to eliminate redundant nodes, resulting in the minimized MTBDD, as illustrated in Figure 4(d). Through merging, the number of nodes in the BDD can be reduced from 13 to 8.

Multi-value Refresh: The idea of multi-value blind rotation, as introduced in [47], enables the evaluation of multiple TableT at the cost of a single blind rotation. The blind rotation during the Refresh step can be expressed as

$$\begin{aligned}\mathsf{TV}(x) \cdot x^{\mathsf{LWE}(m)} &= \mathsf{RLWE}(\sum_{j=0}^{N'} \sum_{i=0}^{\ell-1} \sum_{k=0}^{d-1} \frac{1}{\mathsf{Bg}^k} t_i x^{j+N/\ell \cdot i} \cdot x^m) \\ &= \mathsf{RLWE}((\sum_{j=0}^{N'} \sum_{k=0}^{d-1} \frac{1}{\mathsf{Bg}^k} x^i \cdot x^m) \cdot \sum_{i=0}^{\ell-1} t_i x^{N/\ell \cdot i}.\end{aligned} \quad (12)$$

where $N' = N/(d-1)(\ell-1)$. Here, $\sum_{j=0}^{N'} \sum_{k=0}^{d-1} \frac{1}{\mathsf{Bg}^k} x^i \cdot x^m$ can be considered as a common rotation factor, since it is independent of TableT. When a single blind rotation is used to rotate multiple TableT, the result of the common factor rotation is multiplied by several different $\sum_{i=0}^{\ell-1} t_i x^{N/\ell \cdot i}$. Compared to the rotation process, polynomial multiplication requires significantly less time. When applying multi-value blind rotation during the Refresh step, it is necessary to reconstruct the TableT. For single-bit outputs, TableT only contains two possible values (e.g., $0, 1$). However, in the case of multi-bit outputs, the structure of TableT becomes more complex which needs to enumerate all possible output combinations. For instance, when the number of output bits is two, TableT becomes $\{0, 1, 0, 1\}$ and $\{0, 1, 0, 1\}$, where each

Algorithm 1: Heuristic LUT Merging

Input : A DAG $G = (V, E)$ representing the circuit
Input : Similarity matrix $\mathcal{W}(V_i, V_j)$ for nodes $V_i, V_j \in V$
Input : Function $\mathcal{A}(V_i)$ calculates the area of LUT V_i
Input : Maximum input size $\mathcal{M}_{in}$ and maximum output size $\mathcal{M}_{out}$
Output: DAG $G' = (V', E')$ after merging

1 $T \leftarrow$ Topological order of nodes in V
2 **while** *valid merge candidates exist* **do**
3 $\mathcal{S} \leftarrow$ find a candidate subset in V according to $\mathcal{W}(V_i, V_j)$
4 $I(\mathcal{S}) \leftarrow \bigcup_{i \in \mathcal{S}} I(V_i)$
5 **if** $|I(\mathcal{S})| \leq \mathcal{M}_{in}$ *and* $|\mathcal{S}| \leq \mathcal{M}_{out}$ **then**
6 $V_k \leftarrow \bigoplus_{i \in \mathcal{S}} V_i$
7 Calculate $\mathcal{A}(V_{\mathscr{K}})$
8 **if** $\mathcal{A}(V_{\mathscr{K}}) < \sum_{i \in S} \mathcal{A}(V_i)$ **then**
9 $merge_candidate \leftarrow \mathcal{S}$
10 **if** $merge_candidate$ *found* **then**
11 Merge nodes in $merge_candidate$ into new node $\mathscr{K}$
12 Update V: remove nodes in $merge_candidate$, add $\mathscr{K}$
13 **if** *cycle detected in* G **then**
14 Undo merge of $merge_candidate$
15 **if** *no further reduction in* $\mathcal{A}$ **then**
16 Break the loop
17 **return** *Circuit* G'

element represents a different input-output combination. Note that the size of `TableT` increases as the number of outputs increases.

With the two optimizations, we are able to achieve a more efficient multi-value LUT evaluation, as illustrated in Figure 5. The computation of the `CMUX` tree is transformed into a multi-terminal `CMUX` tree with a smaller node count, and the number of blind rotations in the `Refresh` step is reduced to $1/l$ of the original. The evaluation time of the multi-output LUT becomes:

$$T_{\texttt{M-HomLUT}} = k \cdot T_{\text{conv}} + m \cdot T_{\texttt{CMUX}} + T'_{\texttt{Refresh}},$$

where m denotes the number of `CMUX` in the `CMUX` tree and $T'_{\texttt{Refresh}}$ represents the time spent rotating a larger `TableT`.

*2) **LUT merging heuristics**:* The circuit output after LUT mapping only consists of single-output LUTs. To further reduce computation latency, we aim to merge multiple LUTs into multi-output LUTs. LUTs eligible for merging should satisfy the following two conditions:

① The newly generated LUT must have inputs and outputs that are smaller than the maximum allowed number $\mathcal{M}_{\text{in}}$ and $\mathcal{M}_{\text{out}}$ specified by the given parameters.
② Merging LUTs must not introduce cycles into the circuit, ensuring the new logic structure preserves logical correctness and prevents any unintended loops.

Our goal is to reduce the total circuit size as well as the number of `CMUX` operations under the condition that these requirements are satisfied.

To begin the optimization process, the circuit is represented as a DAG. We define a DAG $G = (V, E)$, where each node V represents a LUT and the edges E denote the dependencies between LUTs. The truth table is stored in the node attributes by a binary sequence. Next, we employ a heuristic merging approach, as outlined in Algorithm 1, to merge single-output LUTs into multi-output LUTs. We first construct a similarity matrix of all nodes evaluated using `HomLUT`. For a pair of

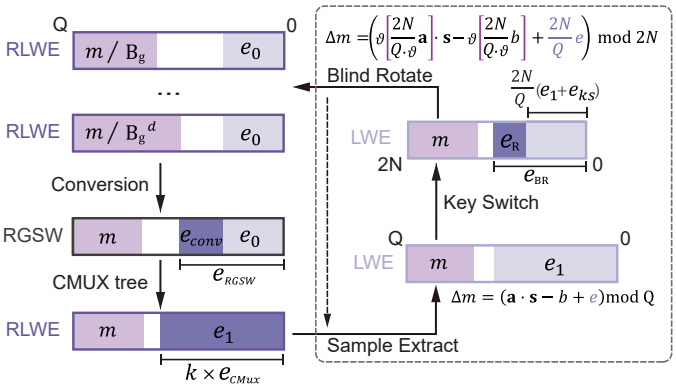

Fig. 6. Variation of noise during a homomorphic LUT evaluation.

nodes V_i and V_j, we align their input sets and extend their `CMUX` trees by introducing don't-care nodes to ensure structural consistency. The similarity $\mathcal{W}(V_i, V_j)$ can be expressed as:

$$\mathcal{W}(V_i, V_j) = \lambda_1 \frac{|I(V_i) \cap I(V_j)|}{|I(V_i) \cup I(V_j)|} + \lambda_2 \frac{|U(V_i) \cap U(V_j)|}{\max(|U(V_i)|, |U(V_j)|)}, \tag{13}$$

where $I(V_i)$ represents the input sets of V_i and $U(V_i)$ represents the set of nodes in the `CMUX` tree representing LUT V_i, with λ_1 and λ_2 as weighting factors satisfying $\lambda_1 + \lambda_2 = 1$. Here, the similarity matrix serves only as a heuristic to prioritize merge candidates for better `CMUX` tree reduction. Next, on Line 1, a topological order of the nodes in V is computed. Then, on Line 3, we identify a candidate set of nodes $\mathcal{S}$, where $|\mathcal{S}| \geq 2$. The selection of nodes is based on the similarity matrix $\mathcal{W}$, prioritizing nodes with higher similarity for potential merging. Then, on Line 4-9, we check whether the subset satisfies the input and output size constraint and whether the node count of the newly merged `CMUX` tree is smaller than the sum of the node counts of the individual `CMUX` trees before merging. If all the conditions are met, we update the candidate merge list to $\mathcal{S}$. On Line 10-14, the nodes in $\mathcal{S}$ are merged into a new node $\mathscr{K}$. The input set of $\mathscr{K}$ is defined as the union of the input sets of all nodes in $\mathcal{S}$, and each output preserves its original Boolean function, with the corresponding truth table recomputed under the unified input set. Specifically, for each original LUT V_i with Boolean function f_i, the corresponding output $F_{\mathscr{K},i}$ satisfies

$$F_{\mathscr{K},i}(x_1, \ldots, x_k) = f_i(x'_1, \ldots, x'_m), \tag{14}$$

where $\{x'_1, \ldots, x'_m\} = I(V_i) \subseteq I(\mathscr{K}) = \{x_1, \ldots, x_k\}$. We then check for any cycles in the updated DAG to ensure that no circular dependencies are introduced. If a cycle is detected, the merge operation is undone. Together, these steps ensure that the merged node remains functionally equivalent to the original. Finally, on Line 15-16, if no further merge candidates can be found, the algorithm terminates, concluding the merging process. The final result is a modified DAG that represents the improved circuit.

V. Parameter Analysis for SALUS

A. Noise Analysis

Since the external product operation is the fundamental building block for most operators, we begin by providing its noise characteristics. For an external product between an RGSW ciphertext C and an RLWE ciphertext, the noise can be expressed as follows:

$$\begin{aligned}\mathrm{Var}(\widetilde{e}_{\mathsf{EP}}) \leq 2dN\left(\frac{\mathsf{Bg}^2+2}{12}\right)\mathrm{Var}(\widetilde{e}_C) \\ +(1+N)\left(\frac{q^2-\mathsf{Bg}^{2d}}{24\mathsf{Bg}^{2d}}+\frac{1}{12}\right)\|C\|_2^2. \end{aligned} \quad (15)$$

The detailed proof is provided in [28]. To enhance clarity in the following discussion, for a given set of gadget decomposition parameters $\{\mathsf{Bg}, d\}$, we define:

$$\mathcal{C}_{\mathrm{err}}(\mathsf{Bg},d)=\left(\frac{\mathsf{Bg}^2+2}{12}\right), \quad (16)$$

and

$$\mathcal{C}_{\mathrm{dec}}(\mathsf{Bg},d)=\left(\frac{q^2-\mathsf{Bg}^{2d}}{24\mathsf{Bg}^{2d}}+\frac{1}{12}\right). \quad (17)$$

Following that, for CMUX with an RGSW ciphertext C and RLWE ciphertexts d_0, d_1, the noise variance is given as follows:

$$\begin{aligned}\mathrm{Var}(\widetilde{e}_{\mathtt{CMUX}}) &= \mathrm{Var}(\widetilde{e}_{c\times(d_0-d_1)+d_1}) \\ &\leq \mathrm{Var}(\widetilde{e}_{c\times(d_0-d_1)}) + \max\left(\mathrm{Var}(\widetilde{e}_{d_0}), \mathrm{Var}(\widetilde{e}_{d_1})\right) \\ &= \mathrm{Var}(\widetilde{e}_{\mathsf{EP}}) + \max\left(\mathrm{Var}(\widetilde{e}_{d_0}), \mathrm{Var}(\widetilde{e}_{d_1})\right). \end{aligned} \quad (18)$$

Since the noise of the ciphertext is refreshed during the Refresh step, the analysis primarily focuses on the noise accumulation within a single HomLUT evaluation cycle. As illustrated in Figure 6, we start with the RLev ciphertext output from the blind rotation in the Refresh step. The noise depends solely on the parameters of the blind rotation process, which is not influenced by the input ciphertext. Blind rotation can be viewed as nCMUX operations, where the inputs of CMUX are fresh ciphertexts. Additionally, the plaintext values of the RGSW ciphertexts are restricted to only 0 and 1. The noise can be expressed as:

$$\mathrm{Var}(\widetilde{e}_0) \leq n(1+N)\mathcal{C}_{\mathrm{dec}}(\mathsf{Bg},d) + 2ndN\sigma^2\mathcal{C}_{\mathrm{err}}(\mathsf{Bg},d) \quad (19)$$

where σ^2 is the variance of the fresh noise.

Next, the RLev ciphertext is converted into a RGSW ciphertext through Conversion. This process consists of three main operations: modular switch (ms), homomorphic trace (tr), and scheme switch (ss). The total noise variance of the conversion process can be expressed as:

$$\begin{aligned}\mathrm{Var}(\widetilde{e}_{\mathrm{conv}}) &\leq \frac{N}{2}\mathrm{Var}(\widetilde{e}_{\mathrm{tr}}) + \mathrm{Var}(\widetilde{e}_{\mathrm{ss}}) + N^2\mathrm{Var}(\widetilde{e}_{\mathrm{ms}}) \\ &= \frac{N^2(N^2-1)}{6}\left(\mathcal{C}_{\mathrm{dec}}(\mathsf{Bg}_{\mathrm{tr}},d_{\mathrm{tr}}) + d\sigma_{\mathrm{tr}}^2\mathcal{C}_{\mathrm{err}}(\mathsf{Bg}_{\mathrm{tr}},d_{\mathrm{tr}})\right) \\ &+ \frac{N(1+N)}{2}\mathcal{C}_{\mathrm{dec}}(\mathsf{Bg}_{\mathrm{ss}},d_{\mathrm{ss}}) + 2dN\sigma_{\mathrm{ss}}^2\mathcal{C}_{\mathrm{err}}(\mathsf{Bg}_{\mathrm{ss}},d_{\mathrm{ss}}) \\ &+ \frac{N^2(N+1)}{12}. \end{aligned} \quad (20)$$

Here, each step is independent and operates with distinct evaluation keys, allowing the decomposition parameters to vary accordingly. More details of the noise analysis at each step are provided in [27].

The RGSW ciphertext is subsequently utilized in the evaluation of the CMUX tree. To simplify the noise analysis, we approximate a k-level CMUX tree as performing k consecutive CMUX operations. Since the k inputs may come from previous steps with different parameter settings, we use $\mathrm{Var}(\widetilde{e}_{\mathsf{RGSW}(i)})$ to denote the ciphertext noise of the ith input. The variance of the noise in the output ciphertext after kCMUX is:

$$\begin{aligned}\mathrm{Var}(\widetilde{e}_1) \leq k(1+N)\mathcal{C}_{\mathrm{dec}}(\mathsf{Bg},d) \\ +\sum_{i=1}^{k} 2d\,\mathrm{Var}(\widetilde{e}_{\mathsf{RGSW}(i)})\mathcal{C}_{\mathrm{err}}(\mathsf{Bg},d). \end{aligned} \quad (21)$$

The sample extract step does not introduce additional noise. The generated LWE ciphertext is subsequently transformed back into an RLev ciphertext through key switching and blind rotation. The noise of the ciphertext input to the blind rotation consists of the original noise from the CMUX tree evaluation and the noise introduced by key switching. Meanwhile, mapping the LWE from $\mathbb{Z}_q$ to $\mathbb{Z}_{2N}$ introduces a new noise due to the rounding process. The total noise input to the blind rotation can be expressed as:

$$\mathrm{Var}(\widetilde{e}_{\mathrm{br}}) = \mathrm{Var}(\widetilde{e}_{\mathrm{R}}) + \frac{2N}{Q}\left(\mathrm{Var}(\widetilde{e}_1) + \mathrm{Var}(\widetilde{e}_{\mathrm{ks}})\right) \quad (22)$$

Specifically, the noise from key switching is

$$\mathrm{Var}(\widetilde{e}_{\mathrm{ks}}) \leq N\mathcal{C}_{\mathrm{dec}}(\mathsf{Bg}_{\mathrm{ks}},d_{\mathrm{ks}}) + dN\sigma_{\mathrm{ks}}^2\mathcal{C}_{\mathrm{err}}(\mathsf{Bg}_{\mathrm{ks}},d_{\mathrm{ks}}). \quad (23)$$

The rounding error is the accumulation of n rounding noises, which can be considered as the sum of n uniformly distributed random variables. We assume a variable $u \sim \mathcal{U}(-\frac{1}{2}\vartheta, \frac{1}{2}\vartheta)$, where ϑ represents the output number of PBSmanyLUT, which, is equal to the decomposition parameter d in this work. Then we obtain:

$$\mathrm{Var}(\widetilde{e}_{\mathrm{R}}) = \sum_{i=0}^{n-1}\mathrm{Var}(u) = \frac{n\vartheta^2}{12}. \quad (24)$$

For the multi-output LUTs, due to the additional polynomial multiplication after blind rotation, the noise changes to

$$\mathrm{Var}(\widetilde{e}_0) \leq \|\mathtt{T}\|_2^2\left(n(1+N)\mathcal{C}_{\mathrm{dec}}(\mathsf{Bg},d) + 2ndN\sigma^2\mathcal{C}_{\mathrm{err}}(\mathsf{Bg},d)\right) \quad (25)$$

where $\|\mathtt{T}\|_2^2$ is the squared norm of TableT, satisfying $\|\mathtt{T}\|_2^2 \leq \mathscr{S}\cdot(\mathcal{Q}-1)^2$. Here, $\mathscr{S}$ is the number of non-zero coefficients, and $\mathcal{Q}$ is the plaintext modulus for the coefficients. Since we are dealing with Boolean computation, this can be simplified to $\|\mathtt{T}\|_2^2 \leq \mathscr{S}$.

As shown in the dashed region of Figure 6, the noise variation in HomGate behaves similarly to the Refresh process. After blind rotation, the output RLWE ciphertext acquires a fixed level of noise. Subsequently, it is converted into an LWE through a sample extract. The LWE ciphertext is then reintroduced into the blind rotation process through key switching. The noise accumulation at each step is detailed earlier in this section.

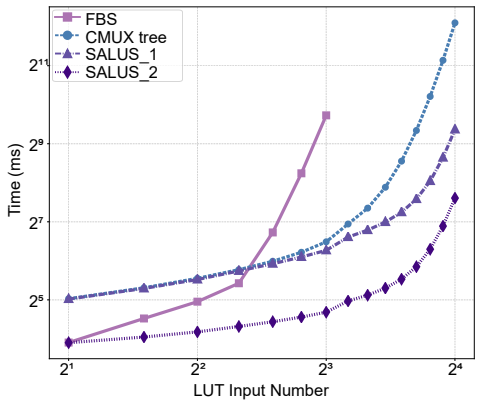

Fig. 7. Performance comparison of LUTs with input sizes ranging from 2 to 16 between SALUS and existing works.

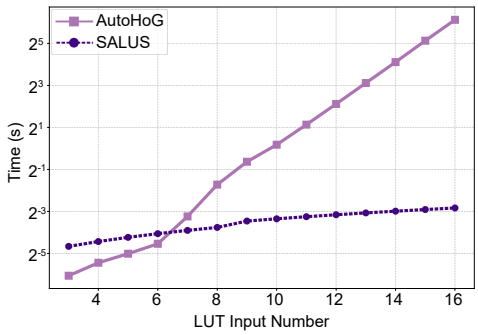

Fig. 8. Performance comparison of the XOR-AND chain function via homomorphic gate in AutoHoG and homomorphic LUT in SALUS.

B. Parameter Setup

We assume the maximum acceptable decryption error probability for the client to be $\mathcal{P}$. The upper bound of $\widetilde{e}_{\text{br}}$ is denoted as $\mathcal{E}$, which satisfies:

$$\mathcal{P} = P(|\widetilde{e}_{\text{br}}| \geq \mathcal{E}) = 2\left(1 - \Phi\left(\frac{\mathcal{E}}{\sqrt{\text{Var}(\widetilde{e}_{\text{br}})}}\right)\right). \quad (26)$$

In order to obtain the correct result in blind rotation, each segment of length $2\mathcal{E}$ in the test vector should have the same value. As a result, the maximum number of such segments is $|\mathtt{T}| \leq \ell_{\max} = \lfloor \frac{N}{2\mathcal{E}} \rfloor$.

The input size constraint $\mathcal{M}_{in}$ for `HomLUT` satisfies two conditions: First, during the `CMUX` tree evaluation, the noise $\widetilde{e}_1$ accumulated through $\mathcal{M}_{in}$ `CMUX` should satisfy $\widetilde{e}_1 = \sum_{i=1}^{\mathcal{M}_{in}} \widetilde{e}_{\mathtt{CMUX}(i)} < \Delta m$, to ensure that the accumulated noise does not overwhelm the message. Second, the noise $\widetilde{e}_{br}$ derived from $\widetilde{e}_1$ should be bounded by $\mathcal{E}$ to maintain correct decryption.

The output size constraint $\mathcal{M}_{\text{out}}$ for `HomLUT` also satisfies two conditions: First, due to the increasing size of `TableT` caused by multi-value blind rotation, this step should respect the size limitation of `TableT`: $2^{\mathcal{M}_{out}} \leq \lfloor \frac{N}{2\mathcal{E}} \rfloor$. Second, the ciphertext noise output from blind rotation should not exceed the decomposed message, i.e., $\widetilde{e}_0 < \frac{\Delta m}{\text{Bg}^d}$.

TABLE I
PARAMETER SETTING AND SECURITY IN DIFFERENT WORKS

Technique	$\log_2 Q$	N	$\log_2 q$	n	$(d, \text{Bg})_{\text{br}}$	(d, Bg) `CMUX`	$\boldsymbol{\lambda}$ **(bit)**
Yu *et al.*	64	2048	32	863	$(1, 2^{23})$	–	≥ 128
Fhewsyn				884	$(1, 2^{23})$	–	
AutoHoG				667	$(3, 2^{6})$	–	
SALUS-1	64	2048	32	571	$(2, 2^{15})$	$(3, 2^{5})$	≥ 128
SALUS-2				643	$(3, 2^{12})$	$(2, 2^{8})$	

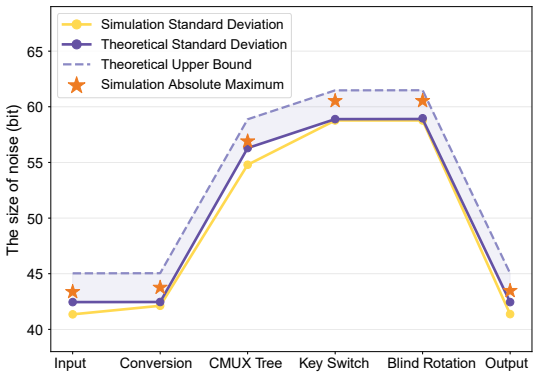

Fig. 9. Empirical noise growth in the HomLUT evaluation pipeline, as observed through Monte Carlo simulations and compared against theoretical estimates.

VI. EXPERIMENT

A. Experiment Setup

We implement SALUS using Python for the synthesis workflow and Rust for the circuit evaluation process. In this paper, we compare the circuit evaluation latency of SALUS with existing works, notably AutoHoG [18], fhewsyn [22], Yu *et al.* [23], Carpov [19] and HELM [24]. All the circuit evaluations are conducted using the TFHE library TFHE-rs [53]. All experiments, except for HELM, are executed on an Intel Xeon Gold 5318Y processor with 512 GB of RAM. The results for HELM are directly referenced from [24].

The parameters instantiated in SALUS and other existing works are summarized in Table I. The parameters are selected to meet 128-bit security based on the LWE-estimator [54]. We set the gadget decomposition parameters for blind rotation key, as well as the RGSW ciphertexts used in the `CMUX` tree, separately. We set the failure probability to $\mathcal{P} = 2^{-32}$, which leads to the following parameter constraints: $\ell_{\max} = 8$, and $\mathcal{M}_{in} = 15$. For the first parameter set (SALUS-1), a single-output LUT is implemented. In the second parameter set (SALUS-2), we set $\mathcal{M}_{out} = 3$. Note that $\mathcal{M}_{in} = 15$ is not a strict upper bound for the number of inputs. However, as the input number increases, the size of the `CMUX` tree grows super-linearly, eventually becoming too large, leading to a decrease in efficiency and an increase in synthesis time. In addition, fhewsyn [22], Yu *et al.* [23], and AutoHoG [18] utilize functional bootstrapping with precisions of 3, 4, and 5 bits, respectively. Carpov [19] and HELM [24] employ a flexible gate size, using different parameters for different circuits, so specific parameters are not provided.

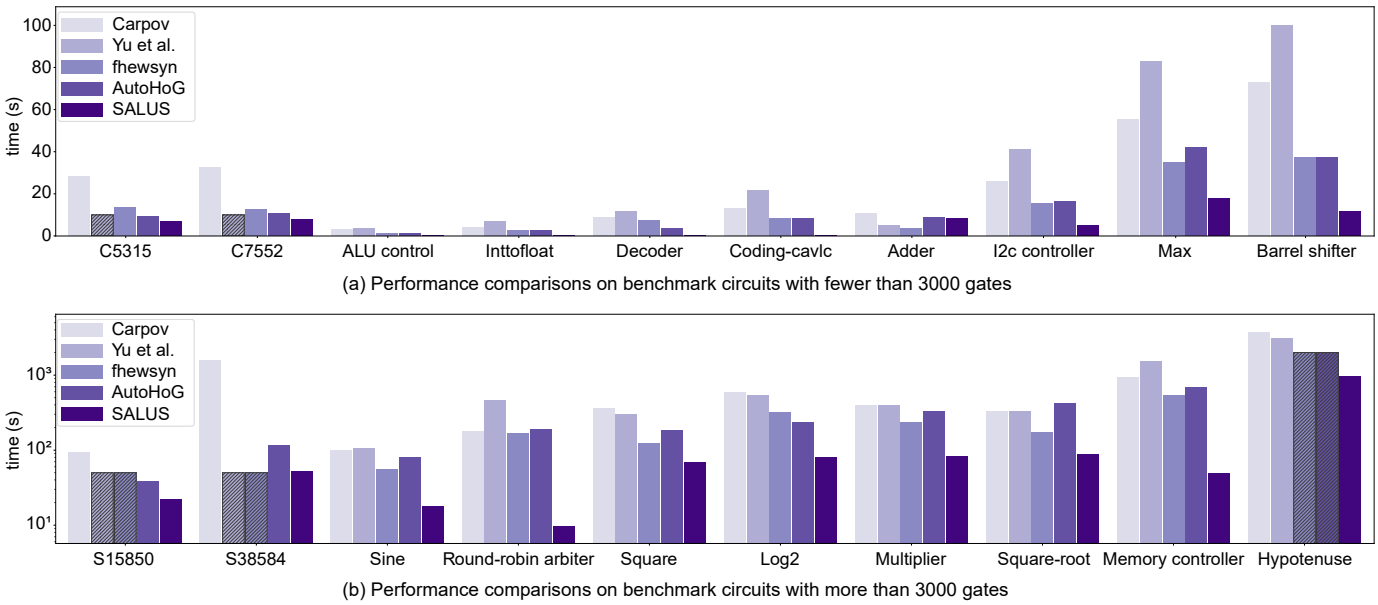

Fig. 10. Performance comparisons on EPFL benchmark circuits and ISCAS benchmark circuits between SALUS and existing works. The bars with diagonal slashes indicate cases where the circuit was either not provided or the synthesis process timed out.

B. Microbenchmarks

Here, we first evaluate the circuit execution latency on a set of microbenchmarks for different truth tables, as summarized in Figure 7. Specifically, for each input size ranging from 2 to 16, we randomly generate 10K truth tables representing different Boolean functions and examine the average evaluation time per output. Each function is evaluated using either a single FBS or a single `CMUX` tree. Notably, the FBS-based evaluation method only supports up to 8-bit precision due to parameter constraints. Compared to FBS-based LUT evaluation, our method achieves up to a $32\times$ speedup. Furthermore, when compared to previous `CMUX`-based approaches [26], we observe up to a $22\times$ reduction in evaluation latency.

We further evaluate a specific l-bit XOR-AND chain function with the form: $f(x_1, x_2, \ldots, x_l) = x_1 \oplus (x_2 \wedge x_3) \oplus (x_4 \wedge x_5) \oplus \cdots \oplus x_l$ with l ranges from 2 to 16. As shown in Figure 8, we compare the evaluation latency using `HomGate` from AutoHoG and the `HomLUT` proposed in SALUS. Here, for bit-widths greater than 8, `HomGate` implementation relies on the radix-based method in TFHE-rs [53]. We observe that when the input size is small (less than 6 bits), `HomGate` has a clear advantage. However, as the input size increases beyond 6 bits, the `HomLUT` outperforms `HomGate`. Additionally, the evaluation time of `HomLUT` grows more gradually, indicating better scalability for larger input sizes. Therefore, in homomorphic circuit, we prefer to use `HomGate` for evaluating small LUTs due to the efficiency at lower input sizes, while leveraging `HomLUT` for large fan-in LUTs to achieve better scalability and overall performance.

To validate the accuracy of our noise analysis, we conduct 10K Monte Carlo simulations and measure the noise at each stage of the `HomLUT` pipeline. As shown in Fig. 9, the `CMUX` tree exhibits the most pronounced noise growth due to the cumulative effect of successive `CMUX` operations, and thus dominates the noise budget, constraining the input and output size of `HomLUT`. After the blind rotation during the `Refresh` step, the noise is reset to the same order of magnitude as the input, enabling the evaluation of arbitrarily deep circuits. Across all stages, the simulated noise standard deviation closely follows the theoretical prediction from our analytical model. Minor deviations between the two trends arise because the analytical model adopts conservative approximations in the noise decomposition. Furthermore, the theoretical upper bound remains strictly above the simulated absolute maximum noise, providing a safety margin for worst-case behavior.

TABLE II
COMPARISON OF SYNTHESIS TIME BETWEEN SALUS AND OTHER HOMOMORPHIC CIRCUIT SYNTHESIS IMPLEMENTATIONS

Benchmarks	Gate#	Implementations	Synthesis Time
Decoder	304	ABC [32]	0.25 s
		fhewsyn [22]	0.79 s
		AutoHoG [18]	2.48 s
		SALUS	1.59 s
Barrel shifter	2952	ABC [32]	0.85 s
		fhewsyn [22]	13.60 s
		AutoHoG [18]	36.99 s
		SALUS	22.96 s
Memory controller	40461	ABC [32]	172.95 s
		fhewsyn [22]	2118.34 s
		AutoHoG [18]	6003.15 s
		SALUS	260.62 s
Hypotenuse	214335	ABC [32]	5060.03 s
		fhewsyn [22]	–
		AutoHoG [18]	–
		SALUS	8706.89 s

C. ISCAS and EPFL Benchmarks

We apply SALUS to a set of standard benchmark circuits, including the arithmetic and random/control benchmark circuits in the EPFL Combinational Benchmark Suite [55],

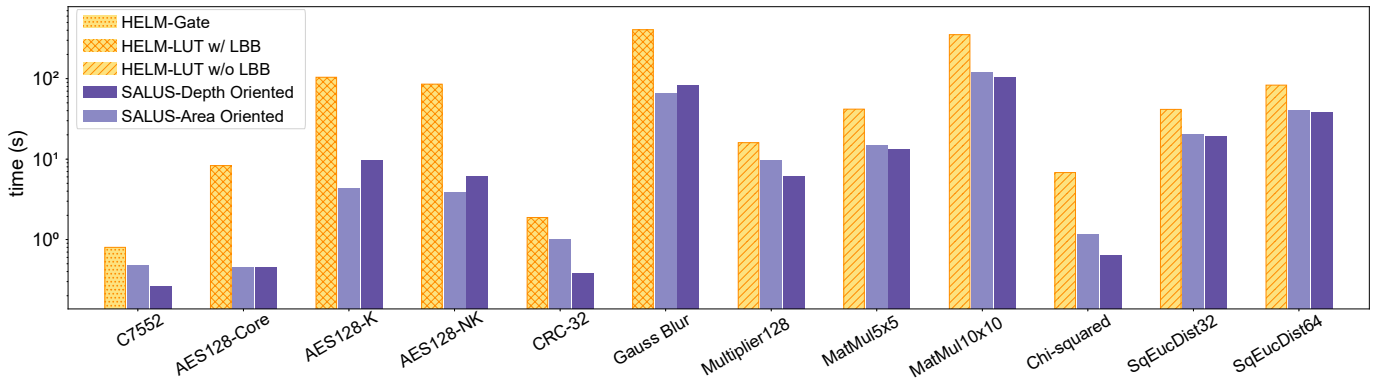

Fig. 11. Performance comparisons on realistic benchmark circuits between SALUS and HELM.

ISCAS'85 benchmark circuits [56] and ISCAS'89 benchmark circuits [57]. For sequential circuits (i.e., ISCAS'89) containing elements such as flip-flops, we adopt the gate re-evaluation technique proposed in [15]. In this approach, an encrypted clock signal is maintained on the server to synchronize state updates. When a flip-flop is encountered, its input is stored for the subsequent cycle, while its output retains the value from the previous cycle. In subsequent cycles, only gates that depend on flip-flops are re-evaluated. The runtimes reported for sequential circuits represent the amortized execution cost per cycle, averaged over ten clock cycles. Overall, we observe that SALUS outperforms existing works on most circuits, with the latency results summarized in Figure 10, achieving up to 26× reduction. The concrete acceleration ratios vary depending on the circuit type. For circuits like `ALU control` and `Decoders`, which inherently exhibit structural redundancies within their LUT implementations, optimization is particularly effective. Moreover, for circuits like `Round-robin arbiter`, mapping via large-fan-in LUTs results in a smaller circuit and overall better performance. In contrast, for circuits like `C5315`, which perform better when mapped to small-fan-in LUTs, the advantage of SALUS becomes less significant. For `Adder` circuit, SALUS exhibits a slowdown compared to fhewsyn, as fhewsyn incorporates a specialized adder substitution optimization. For large-scale circuits with more than 3000 gates, SALUS consistently outperforms the other methods.

As shown in Table II, we compare the synthesis time of different homomorphic circuit synthesis implementations across various benchmarks. ABC [32] serves as the baseline FPGA synthesis tool that performs technology mapping to LUT-15 without specific optimizations for homomorphic evaluation. Fhewsyn [22] primarily relies on Yosys [46] for synthesis and HAL [58] for post-synthesis optimization. AutoHoG [18] applies the gate-replacement and gate-combination techniques proposed in its original work. The synthesis time of SALUS includes both the standard ABC synthesis flow and the additional overhead from LUT merging and per-node BDD reordering. For large-scale circuits, the baseline ABC synthesis accounts for over 60% of the total synthesis time, with the post-synthesis overhead contributing less than 40%. While fhewsyn achieves faster synthesis on small circuits due to its smaller parameter configuration, SALUS outperforms both fhewsyn and AutoHoG as circuit complexity increases. In the case of extremely large circuits such as `Hypotenuse` with over 200,000 gates, SALUS completes synthesis within 3 hours, while fhewsyn and AutoHoG were unable to complete the synthesis process within 12 hours.

TABLE III
COMPARISON OF MEMORY CONSUMPTION BETWEEN SALUS AND HELM

Benchmarks	Gate#	Implementations	Node#/Lev	Memory Usage
C7552	1.3K	HELM-Gate [24]	1.1K/31	97.5 MB
		SALUS	212/3	243 MB
AES Core	19.8K	HELM-LUT [24]	14.7K/18	791 MB
		SALUS	442/5	266 MB
SqEucDist32	93.4K	HELM-Arith [24]	95*/9	735 MB
		SALUS	19.5K/11	1.23 GB
Gauss Blur	1.28M	HELM-LUT [24]	553.7K/26	28.41 GB
		SALUS	92.4K/7	15.17 GB

* In HELM arithmetic mode, node count refers to high-level operations (e.g., add, multiplication) rather than homomorphic Boolean operators.

D. Real-world Benchmarks

In this section, we evaluate large workloads with benchmark designs derived from HELM [24]. Specifically, we consider the following circuits: `C7552` from ISCAS'85 [56]; AES core algorithm and 128-bit AES decryption with and without key scheduling (`AES Core`, `AES128-K`, `AES128-NK`); 32-bit Cyclic Redundancy Check (`CRC-32`); 3 × 3 image filter Gaussian blur (`GaussBlur`); 128-bit multiplier (`Multiplier128`); matrix multiplication across square 5 × 5 and 10×10 matrices with 16-bit unsigned integers (`MatMul5×5`, `MatMul10×10`); 32-bit chi-squared computation (`Chi-Squared`); and 32-bit Squared Euclidean distance with point dimensions of 32 and 64 (`SqEucDist32`, `SqEucDist64`).

HELM [24] exploits inherent parallelism in the circuits through 48 virtual cores to evaluate the encrypted programs. For a fair comparison, SALUS is deployed on 48 virtual cores. Given the consideration of parallelism, we conduct experiments using both area-oriented and depth-oriented LUT mapping strategies during the technology mapping step. As

shown in Figure 11, HELM [24] provides three different modes, and SALUS consistently outperforms the fastest mode for each circuit. For circuits predominantly involving Boolean operations, such as `C7552`, SALUS achieves a $3\times$ speedup over HELM gate mode. For general-purpose scenarios like `AES` and `CRC-32`, where both approaches rely on LUT-based computation, SALUS delivers $5\times$–$24\times$ faster. For circuits primarily composed of arithmetic operations, the arithmetic mode of HELM leverages word-wise addition and multiplication, whereas SALUS does not employ specific optimizations for word-wise computations. Nevertheless, we still achieve $2\times$–$11\times$ acceleration on arithmetic circuits. In most cases, depth-oriented LUT mapping achieves better performance. However, for certain circuits like `GaussBlur`, depth optimization leads to a significant increase in area, which in turn reduces evaluation efficiency.

Table III compares the memory consumption of SALUS and HELM. The two implementations exhibit different memory characteristics due to their distinct circuit representations. In SALUS, each node requires larger truth table storage, and the RLev ciphertext format produces larger intermediate ciphertexts than the LWE ciphertext used in HELM. As a result, SALUS shows higher memory usage on small circuits such as `C7552`. On the other hand, for arithmetic circuits such as `SqEucDist32`, HELM benefits from its arithmetic mode, which processes word-level integers instead of single-bit ciphertexts, resulting in a substantial memory advantage. However, the circuit reduction performed in SALUS partially offsets its ciphertext overhead. For large circuits such as `Gauss Blur`, this reduction enables SALUS achieves lower overall memory consumption compared to HELM.

E. Discussions

While SALUS significantly improves the efficiency of homomorphic circuit evaluation, we also acknowledge several limitations that come as the costs of our optimizations.

Parameter Constraints: SALUS provides operator constraints under a fixed parameter setting that achieves the standard 128-bit security level. If higher security levels are required, the parameter configuration must be re-evaluated, which may further restrict the set of supported Boolean operators. We also note that the maximum LUT input size $\mathcal{M}_{\text{in}} = 15$ is not a strict theoretical upper bound under our parameter setting. In practice, LUTs with 32 or more inputs can be evaluated. However, the size of the CMUX tree grows exponentially with the number of inputs, leading to significantly longer evaluation time and limited practical benefit. Additionally, synthesis time increases sharply because BDD reordering becomes more expensive. Thus, choosing $\mathcal{M}_{\text{in}} = 15$ represents a practical trade-off between LUT evaluation efficiency and synthesis overhead.

Performance on Specific Circuits: SALUS does not include specialized optimizations for common arithmetic components such as adders and multipliers. This results in suboptimal performance on benchmarks such as the 128-bit Adder, compared to fhewsyn [22], which applies dedicated adder substitution strategies. These component-specific optimizations are independent of our design philosophy and can be incorporated into SALUS as future extensions.

Memory Overheads: As discussed in Section VI-D, SALUS incurs high memory overhead for LUT evaluation due to the storage of large truth tables and the larger intermediate ciphertexts produced by its ciphertext format. Although circuit reduction partially offsets this cost, SALUS remains less efficient in memory on small-scale circuits and arithmetic workloads. There remains substantial room for further memory optimizations, such as exploring alternative LUT encoding schemes, or reducing intermediate ciphertext usage through circuit-level and parallelism-aware optimizations.

VII. Conclusion

In this work, we introduce SALUS, a logic synthesis framework for large-scale homomorphic circuits. By leveraging the BDD reordering method and multi-value refresh techniques for homomorphic LUT optimization, along with LUT merging heuristics to utilize multi-output LUTs, we can efficiently evaluate homomorphic circuits derived from a given HDL. We show that SALUS outperforms existing homomorphic circuit synthesis methods on most benchmark circuits, achieving a maximum speedup of $26\times$ in evaluation time.

References

[1] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive,*, 2012.

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

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

[4] N. P. Smart and F. Vercauteren, "Fully homomorphic simd operations," *DESIGN CODE CRYPTOGR.*, vol. 71, pp. 57–81, 2012.

[5] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *EUROCRYPT*. Springer, 2015, pp. 617–640.

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

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

[8] D. Trama, P.-E. Clet, A. Boudguiga, R. Sirdey, and N. Ye, "Designing a general-purpose 8-bit (t) fhe processor abstraction," *Cryptology ePrint Archive*, 2024.

[9] T. Balenbois, J.-B. Orfila, and N. Smart, "Trivial transciphering with trivium and tfhe," in *Proc. WAHC*, 2023, pp. 69–78.

[10] D. Trama, P.-E. Clet, A. Boudguiga, and R. Sirdey, "A homomorphic aes evaluation in less than 30 seconds by means of tfhe," in *Proc. WAHC*, 2023, pp. 79–90.

[11] B. Wei, X. Lu, R. Wang, K. Liu, Z. Li, and K. Wang, "Thunderbird: Efficient homomorphic evaluation of symmetric ciphers in 3gpp by combining two modes of tfhe," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 3, pp. 530–573, 2024.

[12] S. Bian, Z. Zhang, H. Pan, R. Mao, Z. Zhao, Y. Jin, and Z. Guan, "He3db: An efficient and elastic encrypted database via arithmetic-and-logic fully homomorphic encryption," in *CCS*, 2023, pp. 2930–2944.

[13] Z. Zhang, S. Bian, Z. Zhao, R. Mao, H. Zhou, J. Hua, Y. Jin, and Z. Guan, "Arcedb: An arbitrary-precision encrypted database via (amortized) modular homomorphic encryption," in *CCS*, 2024, pp. 4613–4627.

[14] S. Bian, M. Hiromoto, and T. Sato, "Darl: Dynamic parameter adjustment for lwe-based secure inference," in *DATE*, 2019, pp. 1739–1744.

[15] C. Gouert and N. G. Tsoutsos, "Romeo: conversion and evaluation of hdl designs in the encrypted domain," in *DAC*, 2020, pp. 1–6.

[16] S. Gorantala, R. Springer, S. Purser-Haskell, W. Lam, R. J. Wilson, A. Ali, E. P. Astor, I. Zukerman, S. Ruth, C. Dibak, P. Schoppmann, S. Kulankhina, A. Forget, D. Marn, C. Tew, R. Misoczki, B. Guillen, X. Ye, D. Kraft, D. Desfontaines, A. Krishnamurthy, M. Guevara, I. M. Perera, Y. Sushko, and B. Gipson, "A general purpose transpiler for fully homomorphic encryption," *Cryptology ePrint Archive*, 2021.

[17] K. Matsuoka, Y. Hoshizuki, T. Sato, and S. Bian, "Towards better standard cell library: Optimizing compound logic gates for tfhe," in *WAHC*, 2021, pp. 63–68.

[18] Z. Guan, R. Mao, Q. Zhang, Z. Zhang, Z. Zhao, and S. Bian, "Autohog: Automating homomorphic gate design for large-scale logic circuit evaluation," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2024.

[19] S. Carpov, "A fast heuristic for mapping boolean circuits to functional bootstrapping," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 4, pp. 621–643, 2025.

[20] A. Singh, S. Das, A. Chakraborty, R. Sadhukhan, A. Chatterjee, and D. Mukhopadhyay, "Fheda: efficient circuit synthesis with reduced bootstrapping for torus fhe," in *EuroS&P*, 2024, pp. 841–859.

[21] N. Bon, D. Pointcheval, and M. Rivain, "Optimized homomorphic evaluation of boolean functions," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 3, pp. 302–341, 2024.

[22] J. Mono, K. Kluczniak, and T. Güneysu, "Improved circuit synthesis with multi-value bootstrapping for fhew-like schemes," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 4, pp. 633–656, 2024.

[23] M. Yu, S. Carpov, A. Tempia Calvino, and G. De Micheli, "On the synthesis of high-performance homomorphic boolean circuits," in *WAHC*, 2023, pp. 51–63.

[24] C. Gouert, D. Mouris, and N. G. Tsoutsos, "Helm: Navigating homomorphic encryption through gates and lookup tables," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 2822–2835, 2025.

[25] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *ASIACRYPT*. Springer, 2016, pp. 3–33.

[26] R. Wang, Y. Wen, Z. Li, X. Lu, B. Wei, K. Liu, and K. Wang, "Circuit bootstrapping: faster and smaller," in *EUROCRYPT*. Springer, 2024, pp. 342–372.

[27] R. Wang, J. Ha, X. Shen, X. Lu, C. Chen, K. Wang, and J. Lee, "Refined tfhe leveled homomorphic evaluation and its application," in *CCS*, 2025, pp. 2309–2323.

[28] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe," in *ASIACRYPT*. Springer, 2021, pp. 670–699.

[29] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient fpga mapping solution," in *FPGA*, 1999, pp. 29–35.

[30] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 13, no. 1, pp. 1–12, 1994.

[31] J. Cong and Y.-Y. Hwang, "Simultaneous depth and area minimization in lut-based fpga mapping," in *FPGA*, 1995, pp. 68–74.

[32] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *CAV*. Springer, 2010, pp. 24–40.

[33] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *ICCAD*, 2007, pp. 354–361.

[34] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2894–2903, 2006.

[35] F.-X. Reichl, F. Slivovsky, and S. Szeider, "Circuit minimization with qbf-based exact synthesis," in *AAAI*, vol. 37, no. 4, 2023, pp. 4087–4094.

[36] ——, "eslim: Circuit minimization with sat based local improvement," in *SAT*, 2024, pp. 23–1.

[37] A. Costamagna, A. T. Calvino, A. Mishchenko, and G. De Micheli, "Area-oriented resubstitution for networks of look-up tables," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2025.

[38] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "Lsoracle: A logic synthesis framework driven by artificial intelligence," in *ICCAD*, 2019, pp. 1–6.

[39] X. Li, L. Chen, F. Yang, M. Yuan, H. Yan, and Y. Wan, "Himap: a heuristic and iterative logic synthesis approach," in *DAC*, 2022, pp. 415–420.

[40] O. Martinello, F. S. Marques, R. P. Ribas, and A. I. Reis, "KL-cuts: a new approach for logic synthesis targeting multiple output blocks," in *DATE*, 2010, pp. 777–782.

[41] F. Wang, L. Zhu, J. Zhang, L. Li, Y. Zhang, and G. Luo, "Dual-output lut merging during fpga technology mapping," in *ICCAD*, 2020, pp. 1–9.

[42] A. T. Calvino and G. De Micheli, "Technology mapping using multi-output library cells," in *ICCAD*, 2023, pp. 1–9.

[43] C. Yang, V. Singhal, and M. Ciesielski, "BDD decomposition for efficient logic synthesis," in *ICCD*, 1999, pp. 626–631.

[44] B. Bollig and I. Wegener, "Improving the variable ordering of obdds is np-complete," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, 1996.

[45] D. Fried, L. M. Tabajara, and M. Y. Vardi, "BDD-based boolean functional synthesis," in *CAV*, 2016, pp. 402–421.

[46] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/, 2016.

[47] S. Carpov, M. Izabachène, and V. Mollimard, "New techniques for multi-value input homomorphic evaluation and applications," in *CT-RSA*, 2019, pp. 106–126.

[48] A. Mishchenko, S. Chatterjee, R. K. Brayton, and N. Eén, "Improvements to combinational equivalence checking," in *ICCAD*, S. Hassoun, Ed. ACM, 2006, pp. 836–843.

[49] S. Bian, Z. Zhao, Z. Zhang, R. Mao, K. Suenaga, Y. Jin, Z. Guan, and J. Liu, "Heir: A unified representation for cross-scheme compilation of fully homomorphic computation," in *NDSS*, 2024.

[50] S. Bian, Z. Zhao, R. Shen, Z. Zhang, R. Mao, D. Li, Y. Liu, M. Waga, K. Suenaga, Z. Guan *et al.*, "Chloe: Loop transformation over fully homomorphic encryption via multi-level vectorization and control-path reduction," in *IEEE Symp. Security Privacy (SP)*, 2025, pp. 35–35.

[51] J. Katz and Y. Lindell, *Introduction to Modern Cryptography. 2nd ed.* CRC Press, 2014.

[52] F. Somenzi, "Cudd: Cu decision diagram package release 2.3. 0," *University of Colorado at Boulder*, vol. 621, 1998.

[53] Zama, "TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data," 2022, https://github.com/zama-ai/tfhe-rs.

[54] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, pp. 169–203, 2015.

[55] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IWLS*, 2015.

[56] F. Brglez, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran," in *ISCAS*, 1985.

[57] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *ISCAS*. IEEE, 1989, pp. 1929–1934.

[58] Embedded Security Group, "HAL - The Hardware Analyzer," https://github.com/emsec/hal, 2019.



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.



Zhou Zhang received his B.S. in Beihang University, China, in 2021, and is now pursuing the Ph.D. 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.



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



Peng Yin received his B.S. (2013) and M.S. (2017) from Beijing Institute of Technology and his Ph.D. in Cyberspace Security (2025) from the University of Chinese Academy of Sciences. He is currently a Research Fellow with the Woodpecker Innovation Center. His research interests include electromagnetic security in cyberspace and cybersecurity for cyber-physical systems.



Song Bian (Member, IEEE) is currently an associate professor at Beihang University. 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, and served as an assistant professor at Kyoto University until 2021. His main areas of interest include fully homomorphic encryption, privacy-preserving computing and domain-specific hardware accelerator. His publications cover top-tier journals/conferences such as IEEE S&P, USENIX Security, ACM CCS, NDSS, HotChips, IEEE TIFS, IEEE JSSC, etc. He received the Distinguished Paper Award at CCS'23. He served as technical committee members/reviewers for top-tier international conferences/journals across different fields of study, including ACM CCS, USENIX Security, IEEE TIFS and IEEE TCAD. He is a member of ACM and IEEE.

APPENDIX

We summarize the notations and operators used in this work in Table A1.

Appendix Table A1
SUMMARY OF NOTATIONS

Notation	Description
$\mathbf{a}$	An element in vector domain
$\mathbf{a}_i$	The i-th element of $\mathbf{a}$
$\tilde{a}$	An element in polynomial ring
$\tilde{a}_i$	The i-th coefficient of $\tilde{a}$
$\tilde{A}$	An element in matrix domain
$\tilde{A}_i$	The i-th row of $\tilde{A}$
$\tilde{A}_{i,j}$	The element of A in the i-th row, j-th column
$\|\boldsymbol{a}\|_i$	The i-th norm of $\boldsymbol{a}$
λ	The security parameter
p	The plaintext modulus
q	The ciphertext modulus for LWE
Q	The ciphertext modulus for RLWE and RGSW
n	The lattice dimension for LWE
N	The lattice dimension for an RLWE and RGSW
H	The gadget matrix of RGSW
I	The identity matrix.
d	The dimension of H
Bg	The decompose radix of RGSW
$\mathbb{Z}_q^n$	The set of n-vectors over $\mathbb{Z}_q$
R_N	The cyclotomic ring $\mathbb{Z}[X]/(X^N+1)$
$R_{N,Q}$	The cyclotomic ring $\mathbb{Z}_{\mathbb{Q}}[X]/(X^N+1)$
χ	The noise distribution
Δ	The scaling factor
$\boldsymbol{s}$	The secret key for LWE
$\widetilde{s}$	The secret key for the RLWE and RGSW
Δ	The scaling factor
χ_σ	A Gaussian distribution $\mathcal{N}(0,\sigma)$
σ	The fresh noise where $\sigma \sim \chi_\sigma$.
$\mathsf{LWE}_{\boldsymbol{s}}^{n,q}(m)$	An LWE ciphertext encrypting m with parameters (n,q) and key $\boldsymbol{s}$
$\mathsf{RLWE}_{\widetilde{s}}^{N,Q}(\widetilde{m})$	An RLWE ciphertext encrypting $\widetilde{m}$ with parameters (N,Q) and key $\widetilde{s}$
$\mathsf{RLev}_{\widetilde{s}}^{N,Q,\mathsf{Bg},d}(m)$	A tuple of RLWE ciphertexts encrypting decomposed m with parameters (N,Q,Bg,d) and key $\widetilde{s}$
$\mathsf{RGSW}_{\widetilde{s}}^{n,q,\mathsf{Bg},d}(\widetilde{m})$	An RGSW ciphertext encrypting $\widetilde{m}$ with parameters (N,Q,Bg,d) and key $\widetilde{s}$
$\mathcal{P}$	The failure probability
$\mathcal{E}$	The upper bound of the ciphertext noise
$\mathsf{TV}(x)$	The plaintext polynomial for blind rotation
$\mathtt{T}$	The truth table use to generate $\mathsf{TV}(x)$
$\mathscr{S}$	The number of non-zero coefficients of a polynomial
$\mathcal{Q}$	The plaintext modulus for the coefficients of a polynomial
$\ell_{\max}$	The maximum number of elements in $\mathtt{T}$
$\mathcal{M}_{in}$	The maximum number of input in a LUT
$\mathcal{M}_{out}$	The maximum number of output in a LUT