An Efficient Circuit Synthesis Framework for TFHE via Convex Sub-graph Optimization

Animesh Singh[1], Ayantika Chatterjee[1], Anupam Chattopadhyay[2], and Debdeep Mukhopadhyay[1]

[1]Indian Institute of Technology, Kharagpur
[2]Nanyang Technological University (NTU), Singapore

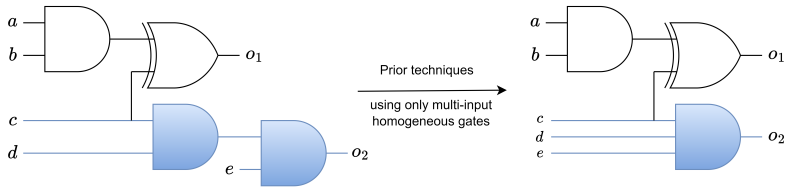
July 29, 2025

Abstract

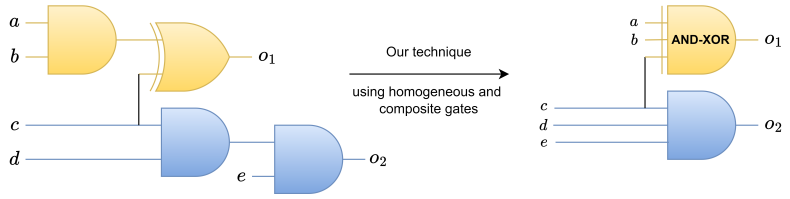
Optimizing Boolean circuits presents a considerable challenge, especially when aiming to construct circuits amenable to Fully Homomorphic Encryption (FHE) schemes. FHE enables arbitrary computations on encrypted data but incorporates a computationally intensive operation called *bootstrapping*, necessary for reducing noise in ciphertexts to facilitate computations on circuits of arbitrary depth. This operation can consume a substantial amount of time, depending on the size of the circuits. To address this issue, we propose a technique for efficiently synthesizing circuits specific to FHE by utilizing multi-input homogeneous and composite Boolean gates. Following this we develop an automated framework for designing efficient circuits compatible with FHE schemes. In this work, we use Torus-FHE (TFHE) (JoC 2019), a widely used FHE scheme for Boolean circuits due to its fast bootstrapping operation per bit. Existing techniques typically employ either multi-input homogeneous gates or, multi-bit Look-Up tables during circuit synthesis, which often limits their ability to produce highly optimized circuits for FHE. Our approach addresses this limitation by proposing viable multi-input composite gates alongwith the homogeneous gates during circuit synthesis. Additionally, we propose an efficient and lightweight circuit synthesis approach based on graph optimization. Our approach identifies convex sub-graphs in a Directed Acyclic Graph (DAG) representing the input circuit and replaces them with a more compact structure. This results in a reduction of the number of nodes in the DAG and so as the number of Boolean gates in the input circuit. Our proposed framework provides the most efficient Boolean circuits for TFHE till date, achieving up to a 20% improvement in homomorphic evaluation time compared to the state-of-the-art general compiler optimization techniques for TFHE, and it also demonstrates a 4-6× improvement over prior work on FHEW-like schemes.

1 Introduction

Fully Homomorphic Encryption (FHE). The advent of FHE has brought a revolution to the cloud computing scenario due to their innate ability to perform computation directly over encrypted data without the need to decrypt it first. However, most



(a) Example circuit optimization using prior techniques.



(b) Example circuit optimization using our technique.

Figure 1: Comparison between previous techniques and our proposed method on an example circuit shows that our framework employs a composite AND-XOR gate, decreasing the number of bootstrapping operations required to evaluate the circuit.

of the FHE schemes [Chillotti et al.(2020a), Ducas and Micciancio(2015), Brakerski(2012)] inherently suffer from performance bottleneck due to the presence of a costly *bootstrapping* [Gentry(2009)] operation. The reason is that the security of the mathematically hard problems [Regev(2009), Lyubashevsky et al.(2010)] underlying these schemes depends on the presence of *noises*, generally sampled from Gaussian distributions. These noises, added during encryption, grow with each homomorphic operation performed on the FHE ciphertexts. The correctness of the (decrypted) result cannot be guaranteed once the overall noise crosses a pre-defined threshold. To prevent this, most FHE schemes use bootstrapping to decrypt away the accumulated noises homomorphically, thus resetting their level in the resultant ciphertext. However, given this decryption is done homomorphically using encryption of the secret key, it is much slower to perform than decryption in the clear. Particularly, FHEW [Ducas and Micciancio(2015)] and TFHE [Chillotti et al.(2020a)] are widely used FHE schemes designed for Boolean circuits having faster bootstrapping. Although, TFHE [Chillotti et al.(2020a)] achieves the fastest bootstrapping per binary gate evaluation on ciphertexts, its computation time remains significantly slower than that of plaintext operations. Therefore, the challenge is to evaluate Boolean circuits efficiently on encrypted inputs using TFHE. However, the general circuit synthesis tools such as Cadence Genus[1], Yosys [Wolf(2016)], etc implement algorithms that synthesize Boolean circuits, especially for area/depth optimizations, which are irrelevant for generating efficient circuits for TFHE computations, because TFHE-friendly circuits have to be optimized in terms of the total bootstrapping operations. Thus optimization techniques for TFHE schemes require reducing the number of gates (that requires bootstrapping) not the area/depth in a circuit.

Several studies [Songhori et al.(2015), Demmler et al.(2015), Ghodsi et al.(2020)] have proposed circuit synthesis methods aimed at optimizing Garbled and MPC circuits with a focus

[1]https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html

on reducing latency and area footprint. However, it is important to note that the optimization strategies used in these approaches for Garbled and MPC circuits do not apply to the FHE domain. These methods incorporate more NAND gates to create circuits optimized for latency, as NAND gates are favored over AND, XOR gates in classical flows due to their smaller area and lower latency. The traditional Boolean circuit optimization techniques mainly focus on reducing the area or depth of a circuit. However, area optimization might increase the number of binary gates by substituting XOR gates (which are considered to occupy more area) with NAND gates (which occupy less area). Since replacing a single XOR gate requires four NAND gates, this could lead to a significant increase in the total number of gates. However, a depth optimization algorithm may increase the number of binary gates at specific levels within the circuit while reducing the overall number of levels[2]. This is in contrast to our proposed method, which prioritizes the use of AND and/or XOR gates during optimization to generate circuits better suited for TFHE. Our present work focuses on TFHE [Chillotti et al.(2020a)], where each binary gate evaluation is followed by a bootstrapping operation. Therefore, to minimize the total number of bootstrapping function calls, a circuit synthesis algorithm should aim to reduce the number of binary gates in the circuit, which contrasts with the goals of area or depth optimization algorithms.

Optimizing TFHE with Parallel Implementations. Over the years, various approaches have been suggested to expedite the evaluation of Boolean circuits in FHE. Some recent studies [Van Beirendonck et al.(2023), Jiang et al.(2022), Hu et al.(2024), Gouert et al.(2023a), Gouert et al.(2023b), Turan et al.(2020)] have focused on reducing bootstrapping time by employing parallel implementations utilizing GPUs and FPGAs. These efforts aim to minimize bootstrapping costs through parallel computations on these hardware accelerators. In contrast, our approach concentrates on circuit synthesis to reduce the frequency of bootstrapping operations during circuit evaluation. These circuits are known to be FHE-amenable circuits. Therefore, our work can provide an additional layer of optimization on the existing hardware accelerators, leveraging the reduction in both bootstrapping cost and count.

General-purpose Optimizations for TFHE. Tools like Concrete[3] and Google Transpiler [Gorantala et al.(2021)] provide general-purpose compilers to convert a C++ or Python code to circuits suitable for FHE computations. While these compilers do not specifically optimize circuits to minimize bootstrapping count, they aim to convert functions into FHE-specific circuits capable of operating on encrypted data. This involves techniques such as loop unrolling, conditional statement replacement with multiplexers, utilizing look-up tables (LUTs) for faster homomorphic computations, etc. In contrast, our objective is to develop a general-purpose optimizer for these FHE-specific circuits by further reducing the number of bootstrapping requirements during evaluation on encrypted data. For example, the homomorphic evaluation of AES-128 circuit using Concrete [Chillotti et al.(2020b)] takes around 9211 secs[4], whereas our optimized representation of AES-128 circuit only takes 411 secs.

In recent years, Matsuoka et al. [Matsuoka et al.(2021a)] proposed an efficient five-stage pipeline processor to improve the performance in circuit evaluation by providing parallelism. They proposed a custom Instruction Set Architecture (ISA) to reduce the gate count of

[2]A level in a Boolean circuit represents a hierarchical step in the circuit's structure, indicating the layers of computation required to progress from the inputs to the output.

[3]https://github.com/zama-ai/concrete

[4]This reported timing comes from straight-forward usage of Concrete without any optimization on Concrete. This result is taken from Table 1 of [Gouert et al.(2023a)].

the processor and CMUX Memory-based ROM and RAM constructions to speed up the instruction evaluation over encrypted data. A recent work by Mono et al. [Mono et al.(2024)] proposes efficient circuit optimization techniques for FHEW [Ducas and Micciancio(2015)] by utilizing multi-bit LUT-based approaches with the Full Adder (FA) circuits and followed by an efficient LUT grouping mechanism to incorporate multi-value functional bootstrapping (FBS) in FHEW [Ducas and Micciancio(2015)]. Their approach significantly reduces the total bootstrapping count while evaluating the circuit on encrypted inputs in FHEW. But, for evaluation correctness, they require a larger parameter set which increases their FBS cost (upto 0.58 secs) and thus increases the homomorphic evaluation time of the overall circuit. Alternatively, our approach is built on TFHE and without requiring any modifications to the original TFHE parameter set, resulting in a bootstrapping time of just 17 ms. This allows us to achieve a 4-6× improvement in circuit evaluation time compared to the work of [Mono et al.(2024)].

A similar approach based on multi-value FBS technique is also adopted by Guan et al. [Guan et al.(2024)] and Yu et al. [Yu et al.(2023)] for TFHE; but they also suffer from a higher FBS cost, thus significantly increasing the overall circuit evaluation time although having lesser number of bootstrapping operations. In Section 4 we provide detailed comparisons with these works. However, the works [Singh et al.(2023), Matsuoka et al.(2021b)] follow a different footprint, focusing on developing efficient standard cell libraries by incorporating multi-input Boolean gates to synthesize efficient circuits specific to TFHE. In particular, Matsuoka et al. [Matsuoka et al.(2021b)] designed efficient Half Adder and Full Adder circuits with a reduced number of blind rotations and key switching, the two primary operations required in bootstrapping. Whereas, Singh et al. [Singh et al.(2023)] presented a circuit optimization technique that uses 3-input homogeneous gates such as 3-input AND, XOR, etc. Their approach utilizes resource-intensive tools like Cadence Genus with augmented cell libraries to re-synthesize a Boolean circuit consisting of only binary gates and transform it into an efficient representation consisting of both 2-input and 3-input homogeneous gates. However, multi-input homogeneous gates can only simplify sequences of identical gates, and they fall short when it comes to optimizing consecutive different gates. Consequently, incorporating multi-input composite gates offers the advantage of reducing sequences of different types of binary gates in a Boolean circuit, leading to more efficient optimization. Therefore, our approach incorporates both homogeneous and viable composite gates to synthesize a Boolean circuit. With multi-input "composite gates" we refer to a combination of two different binary gates with a single bootstrapping function call. For example, a composite AND-XOR gate takes three inputs. The first two inputs are evaluated with an AND operation. The output of AND and the third input are evaluated with an XOR operation. Additionally, we also introduce a 4-input homogeneous gates (AND and OR), whereas the prior works only consider upto a 3-input. In Section 3.1, we define these new multi-input composite and homogeneous gates in more detail.

Furthermore, our circuit synthesis approach is much simpler and lighter compared to the prior works [Singh et al.(2023), Matsuoka et al.(2021b)], because we perform the circuit synthesis by mapping it into a graph optimization problem. In particular, we propose to optimize a convex sub-graph within a Directed Acyclic Graph (DAG), representing a target circuit to integrate a multi-input gate in the circuit; thus eliminating the need for resource-intensive synthesis tools such as Cadence Genus or Yosys. Also, the approaches of augmenting cell libraries to include multi-input gate definitions (especially for multi-input composite gates) introduce additional challenges and require manual oversight. Therefore,

our approach based on convex sub-graph optimization simplifies the circuit synthesis technique while achieving better efficiency than existing works.

Hand-tuned Optimizations for TFHE. Several hand-tuned implementations [Wei et al.(2023), Stracovsky et al.(2022), Trama et al.(2023), Wang et al.(2024)] reported better performance on selective benchmarks. However, [Wei et al.(2023), Stracovsky et al.(2022), Trama et al.(2023)] are very specific to AES circuits or other Feistel structure ciphers, such as SM4. Their primary goal is to use functional bootstrapping (FBS) and multi-threaded implementation to accelerate the sub-operations (AddRoundKey, SubBytes, MixColumns) of AES. Furthermore, [Wang et al.(2024)] presents an efficient approach to perform FBS with a reduced key size.

FBS is most effective for non-linear, multi-input, multi-output operations, as its cost grows super-linearly with fan-in but is nearly independent of fan-out [Carpov(2024)]. In AES evaluation, 8-bit to 8-bit LUTs with FBS are advantageous as the sub-operations of AES work on bytes. However, for circuits with mainly linear, sequential components and low fan-in or fan-out, LUT-based implementations are inefficient, thus reducing overall performance. In Section 4 we presented a detailed discussion on the performance of LUT-based designs and FBS for different structures of circuits.

These techniques also use 16 threads to parallelize AES state matrix operations, extendable to SIMD tasks like convolutions but less effective for sequential circuits like Min/Max. In contrast, In contrast, our goal is to develop a general-purpose optimizer capable of optimizing arbitrary Boolean circuits for TFHE, supporting both high-level C++ and low-level Verilog programs. For comparison, our approach can evaluate an AES-128 circuit in 411 seconds using a single-core implementation. For a more detailed discussion on the scalability of these hand-tuned optimizations for general benchmarks, we refer to Appendix C. Additionally, we provide an overview of scenarios where these techniques can be integrated with our approach to achieve further improvements.

1.1 Our Contribution

This work proposes potential combinations of composite and homogeneous gates and develops an efficient and lightweight circuit synthesis framework that optimizes Boolean circuits for TFHE. Our primary contributions have the following two directions–

Multi-input Composite Gates. Designing multi-input composite gates specifically for TFHE presents challenges due to the scheme's plaintext encoding and the structure of its homomorphic operation. Not every combination is feasible in TFHE; for instance, constructing an XOR-AND gate is not possible (discussed in detail in subsequent sections). Therefore, one of our key contributions in this work is to design feasible multi-input composite gates that are compatible with TFHE and require only one bootstrapping operation. Specifically, combinations such as AND followed by XOR and OR followed by XOR can be merged into equivalent AND-XOR and OR-XOR gates, respectively. Our circuit synthesis framework utilizes these new types of multi-input composite gates, alongwith the multi-input homogeneous gates (from the prior techniques [Singh et al.(2023)]), to generate optimal circuits for TFHE. Furthermore, we also introduce new 4-input versions of Boolean AND and OR operations, where the correctness of three consecutive AND (or, OR) gate evaluations can be maintained with only one bootstrapping procedure. Based on these constructions, we have

developed an automated framework that takes a Boolean circuit comprising only binary gates as input and re-synthesizes the circuit by incorporating multi-input homogeneous and composite gates to reduce the circuit's size in terms of the number of gates.

Circuit Synthesis via Convex Sub-graph Optimization. Our second contribution is to develop an efficient circuit synthesis framework that alleviates the need for resource-intensive synthesis tools like Cadence or Yosys. In this regard, we propose an automated framework for circuit synthesis using a graph optimization technique. In particular, our framework first converts a Boolean circuit into an equivalent Directed Acyclic Graph (DAG) in which the total number of nodes in the DAG is equal to the total number of binary operations present in the input Boolean circuit. A directed edge is inserted between two nodes in the DAG only if the output of a binary gate is used as input to another binary gate, corresponding to the respective nodes. After constructing the DAG, our framework tries to identify convex sub-graphs with DAG. A convex sub-graph is a sub-graph in which there exists no outgoing edge (except the last node in topological order) from the sub-graph to the other parts of the DAG. A convex sub-graph represents a potential "consecutive" structure in the corresponding circuit, which can be combined into a multi-input gate. Thus replacing such convex sub-graphs with a compact structure essentially includes a multi-input gate (homogeneous or composite) in the corresponding circuit. This simplicity makes our approach a more effective alternative for optimizing Boolean circuits for TFHE. In Section 3.3, we present an algorithm to describe our proposed graph optimization technique in greater detail.

To demonstrate the effectiveness of our proposed synthesis approach, we provide a comparison using an example circuit shown in Figure 1. With prior techniques, optimization is limited to consecutive similar gates (highlighted in Figure 1a). In contrast, our circuit synthesis framework allows for further optimization using a composite AND-XOR gate (see Figure 1b), which results in a further reduction of the total Boolean gate count in the circuit and hence total bootstrapping operations, since each Boolean gate in TFHE requires a single bootstrapping.

2 Preliminaries and Background

2.1 Notations and Mathematical Background

$\mathbb{T}$ denotes the Torus ($\mathbb{T} = \mathbb{R} \pmod 1$), and we use $\mathbb{T} = [-\frac{1}{2}, \frac{1}{2})$. Now, if $\phi \geq \frac{1}{2}$, then $\phi \pmod 1 = (\phi - 1)$; similarly if $\phi < -\frac{1}{2}$, then $\phi \pmod 1 = (\phi + 1)$. We write $x \leftarrow \mathcal{X}$ to represent that an element x is sampled uniformly at random from a set/distribution $\mathcal{X}$. For $a, b \in \mathbb{Z}$ such that $a, b \geq 0$, we denote by $[a]$ and $[a, b]$ the set of integers lying between 1 and a (both inclusive), and the set of integers lying between a and b (both inclusive), respectively. We denote $\langle \mathbf{x}, \mathbf{y} \rangle$ to represent a vector dot product between the vectors $\mathbf{x}$ and $\mathbf{y}$. We refer to $\lambda \in \mathbb{N}$ as the security parameter and denote by $poly(\lambda)$ and $negl(\lambda)$ any generic (unspecified) polynomial function and negligible function in λ, respectively.[5]

[5] A function $f : \mathbb{N} \to \mathbb{N}$ is said to be negligible in λ if for every positive polynomial p, $f(\lambda) < 1/p(\lambda)$ when λ is sufficiently large.

2.2 Fully Homomorphic Encryption

Here, we briefly discuss the definition of FHE and its functionality. For example, in an additive homomorphic encryption (AHE) scheme, $\mathsf{AHE.Dec}(\mathsf{AHE.Enc}(m_0{+}m_1)) = \mathsf{AHE.Dec}(\mathsf{AHE.Enc}(m_0))+$

$\mathsf{AHE.Dec}(\mathsf{AHE.Enc}(m_1))$. Fully Homomorphic Encryption (FHE) extends this property to support both addition and multiplication over encrypted data. Modern FHE schemes rely on hardness assumptions like Learning with Errors (LWE) or Ring Learning with Errors (RLWE) to introduce carefully calibrated noise during encryption. However, this noise grows during computation, potentially leading to incorrect decryption once it surpasses a certain threshold. To address this issue, FHE schemes employ ciphertext maintenance operations called bootstrapping that reduce the noise growth without altering the result.

2.3 Torus FHE (TFHE)

Now, we provide a formal definition of TFHE scheme [Chillotti et al.(2020a)], which is the underlying encryption scheme in the present work.

Definition 1 (Torus FHE)**.** *TFHE scheme is a tuple of five PPT (Probabilistic Polynomial Time) algorithms* ($\mathsf{TFHE.Setup}, \mathsf{TFHE.KeyGen}$, $\mathsf{TFHE.Enc}, \mathsf{TFHE.Dec}, \mathsf{TFHE.Eval}$).

- $\mathsf{params} \leftarrow \mathsf{TFHE.Setup}(1^\lambda)$*: Given a security parameter* λ*, this function returns a public parameter* params*.*
- $(\mathbf{sk}, \mathsf{pk}) \leftarrow \mathsf{TFHE.KeyGen}(\mathsf{params})$*: It takes the public parameter* params *as input and generates a secret key* $\mathbf{sk}$ *and the corresponding public key* pk*.*
- $\mathbf{ct} \leftarrow \mathsf{TFHE.Enc}(m, \mathsf{pk})$*: It takes a message* $m \in \{0,1\}$ *and the public key* $\mathsf{pk} = \langle \mathbf{a}, \mathbf{sk} \rangle + e$ *as input. Samples a random vector* $\mathbf{a} \in \mathbb{Z}_q^n$ *and a noise value* $e \leftarrow \mathcal{G}_\alpha$ *and computes* $b = \sum_{\mathcal{S}} \mathsf{pk} + \Delta(m)$*, where,* $\sum_{\mathcal{S}}$ *denotes random subset sum over the set* $\mathcal{S}$*,* $\Delta(m)$ *is the encoding of the plaintext* m *and* $\mathcal{G}_\alpha$ *is a Gaussian distribution with standard deviation* α *(also known as "noise bound"). It finally returns a ciphertexts* $\boldsymbol{ct} = (b, \mathbf{a})$*.*
- $m \leftarrow \mathsf{TFHE.Dec}(\mathbf{ct}^\star, \mathbf{sk})$*: Given an evaluated ciphertext* $\mathbf{ct}^\star = (b^\star, \mathbf{a}^\star) \in \mathbb{T}^{n+1}$ *and the secret key* $\mathbf{sk}$*, it computes a phase* ϕ *as,* $\phi = b^\star - \langle \mathbf{a}^\star, \mathbf{sk} \rangle = \Delta(m^\star) + e^\star$*, here* $m^\star$ *be the plaintext bit of* $\mathbf{ct}^\star$ *and* $e^\star$ *be the accumulated noise in it. Finally, returns a message bit* $m^\star \in \{0,1\}$ *which minimizes* $|b^\star - \langle \mathbf{a}^\star, \mathbf{sk} \rangle - \Delta(m^\star)|$*.*
- $\mathbf{ct}^\star \leftarrow \mathsf{TFHE.Eval}(\{\mathbf{ct}_i\}_{i \in \ell}, \mathsf{pk}, \mathcal{C})$*: It takes a set of ciphertexts* $\{\mathbf{ct}_1, \ldots, \mathbf{ct}_\ell\}$*, the public key* pk *and a circuit* $\mathcal{C} : \{0,1\}^\ell \to \{0,1\}^{\ell'}$*. Returns an evaluated ciphertext* $\mathbf{ct}^\star$ *which is an encryption of* $m^\star = \mathcal{C}(m_1, \ldots, m_\ell)$ *under the secret key* $\mathbf{sk}$*, where* m_l *is the underlying plaintext of* $\mathbf{ct}_l$ *for all* $l \in \ell$*.*

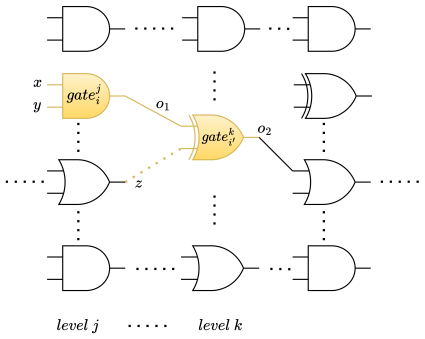
IND-CPA security: The TFHE scheme is IND-CPA secure [Wei et al.(2023)]. For any security parameter $\lambda \in \mathbb{N}$, for any PPT adversary $\mathcal{A}_d$, letting

$$\gamma_\beta = \mathsf{Pr}[\mathsf{G}_{\mathsf{TFHE}, \mathcal{C}_h, \mathbb{A}_d}(1^\lambda) = \beta],$$

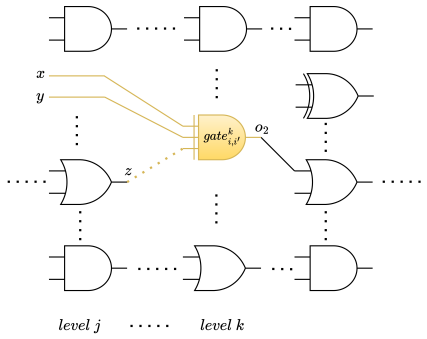
for $\beta \in \{0,1\}$ (where the probability is over the random coins used by TFHE.KeyGen, TFHE.Enc, TFHE.Eval, and the adversary $\mathcal{A}_d$), we have $\left|\gamma_0 - \gamma_1\right| \leq negl(\lambda)$.

3 Proposed Methodology

In this section, we present a detailed discussion on our proposed circuit synthesis flow and provide a comprehensive analysis, highlighting how our approach differs from the circuit synthesis methodologies employed in previous studies [Singh et al.(2023), Matsuoka et al.(2021b)]. Our approach to circuit synthesis begins by converting an input Boolean circuit (consisting of binary gates) into an equivalent directed acyclic graph (DAG) representation. This transformation reduces the circuit synthesis problem to a graph optimization problem, facilitating more efficient optimization. Solving a graph optimization problem requires no additional resources or synthesis tools, and can be computed using a single processing unit. Thus, making it lightweight and efficient in terms of resource-utilization compared to prior works. We describe our proposed circuit synthesis framework in two steps– first, we discuss our proposed 3-input composite gates and 4-input homogeneous gates with their correctness. Second, we provide our circuit synthesis framework using convex sub-graph optimization.



(a) Potential gate structure for 3-input composite gate.



(b) 3-input composite gate reduces the total gate count.

Figure 2: Pictorial description of composite gate structure, that reduces the circuit size.

3.1 Proposed Multi-input Composite Gates

In this section, we illustrate our proposed 3-input composite gates that combine two consecutive but different binary operations. As shown in Figure 2a, the output o_1 of $gate_i^j$ at the j^{th} level is connected as an input to $gate_{i'}^k$, and similarly, the intermediate result o_1 is not used elsewhere in the circuit. Here, i, i' denote the type of gates i.e., AND, XOR, etc. These structures can be combined into 3-input gate, depending on the order of the binary gates in the structures. For example, if an AND gate comes first as $gate_i^j$ and an XOR gate follows as $gate_{i'}^k$ (see Figure 2a), then this particular structure can be combined into a composite AND-XOR gate that requires only one bootstrapping. However, if an XOR gate comes first and is followed by an AND gate, then within the TFHE framework, we cannot evaluate an XOR-AND gate with just one bootstrapping call, thus eliminating the possibility of such a

combination. In the following paragraphs, we present a detailed analysis and proof of the specific order of binary gates required to construct equivalent 3-input composite gates that are feasible within the TFHE scheme.

Composite Gate Construction. Now, we discuss the plausible order of binary gates required to be present in a consecutive structure such that these structures can be replaced with equivalent 3-input Boolean gates. In the TFHE scheme, every binary gate takes two ciphertexts $\mathbf{ct}_1$ and $\mathbf{ct}_2$ as input and returns a ciphertext after homomorphic evaluation of the binary function. To operate correctly within the TFHE scheme, the encoding (Δ) of the underlying plaintexts of the input ciphertexts should be $\Delta = \frac{1}{8}$. This means, for a plaintext m, $\Delta(m) = -\frac{1}{8}$, for $m = 0$ and $\Delta(m) = \frac{1}{8}$, for $m = 1$. After homomorphic evaluation of the binary function, the plaintext encoding in the resultant ciphertext changes to $> \frac{1}{8}$, i.e., greater than $\frac{1}{8}$. The binary gates defined in TFHE cannot operate on plaintext encoding $> \frac{1}{8}$; they only work on plaintext encoding $\frac{1}{8}$ or $< \frac{1}{8}$. Since TFHE does not support division operations, it is challenging to convert an encoding $> \frac{1}{8}$ to $\leq \frac{1}{8}$. Therefore, after each binary gate, a bootstrapping function call is required not only to reduce the noise level but also to re-scale the plaintext encoding to $\frac{1}{8}$. However, in a 3-input gate, the noise level can be managed by sampling the noise during encryption from a Gaussian distribution with a standard deviation of 2^{-15}, as discussed in the work [Singh et al.(2023)]. With this standard deviation, the noise level after two consecutive binary operations remains within the threshold of decryption correctness, i.e., $\frac{1}{16}$. After addressing the noise parameter, constructing a correct 3-input gate additionally requires that the plaintext encoding within the input ciphertexts be chosen appropriately. As previously discussed, the binary gates defined in the TFHE library work with plaintext encoding $\leq \frac{1}{8}$, except for the XOR gate, which can also operate with plaintext encoding $\leq \frac{1}{4}$. Now, we broadly categorize our consecutive structures with different binary gates and describe the combination techniques using the following theorems.

3-*input composite Boolean gates.* We now define and prove the correctness of our proposed 3-input composite gate combinations, including AND-XOR, OR-XOR, etc.

Theorem 1. *In TFHE, the XOR gate followed by any other binary gate can not be combined into an equivalent n-input version* $\forall n \geq 3$.

Proof. The XOR gate in TFHE is defined as

$$\mathbf{ct}' = \left(\frac{1}{4}, \mathbf{0}\right) + 2 \cdot (\mathbf{ct}_1 + \mathbf{ct}_2),$$

where $\mathbf{ct}_i, \forall i \in \{0, 1\}$ encrypts plaintext $\frac{1}{8}$ if the message is 1, and encrypts plaintext $(-\frac{1}{8})$ if the message is 0. From the definition of XOR, one can observe that the plaintext encoding in the resultant ciphertext $\mathbf{ct}'$ can become $> \frac{1}{8}$. A binary gate other than XOR cannot operate on this modified plaintext encoding. Therefore, any gate combination where a binary gate (except XOR itself) follows an XOR gate cannot be combined into an equivalent 3-input gate. □

Since the XOR gate in TFHE can operate on plaintext encoding $\leq \frac{1}{4}$, the output of a binary gate can be used as input to an XOR gate without requiring a bootstrapping operation on the intermediate result. Therefore, a gate structure composed of a binary gate (AND, OR,

XOR, etc.) followed by an XOR gate can be combined into an equivalent 3-input version. In this work, we define a composite gate as follows–

$$\mathbf{ct}_{out} = Gate_1 - Gate_2(\mathbf{ct}_1, \mathbf{ct}_2, \mathbf{ct}_3),$$

where the output $\mathbf{ct}_{out}$ is computed as $Gate_2\left(Gate_1(\mathbf{ct}_1, \mathbf{ct}_2), \mathbf{ct}_3\right)$. According to the prior works [Singh et al.(2023), Matsuoka et al.(2021b)], to construct a 3-input Boolean gate, we need to reduce plaintext encoding from $\frac{1}{8}$ to $\frac{1}{16}$ i.e., $\Delta(m) = \frac{1}{16}$ if $m = 1$ and $\Delta(m) = -\frac{1}{16}$ if $m = 0$. Note that, TFHE does not support division, hence this reduction in encoding can only be performed before encryption.

It is worth noting that, TFHE [Chillotti et al.(2020a)] provides two noise bounds $\geq \sigma_{min} = 2^{-15}$ to ensure security and $\leq \sigma_{max} = 2^{-6}$ to ensure correctness (refer Table 2 for our parameter set). To prove the correctness of a 3-input gate, [Singh et al.(2023)] suggested avoiding sampling noise from σ_{max}, which can lead to a decryption failure with a non-negligible probability after two consecutive operations (i.e., a 3-input gate evaluation) and to choose noise from $\sigma_{min} = 2^{-15}$ to ensure both security and correctness. Hence, the absolute values of the encryption noises are now $|e| \leq 4 \cdot \sigma_{min} = 2^{-13}$ with a probability more than 99.99% according to the property of Gaussian distributions. Now, with this modified encoding and noise distribution, we prove the constructions of composite gates using the following lemmas.

Lemma 1. *In TFHE, a binary AND gate followed by a binary XOR gate can be combined into an equivalent* 3*-input AND-XOR gate.*

Proof. We first define the structure of our proposed AND-XOR gate as follows,

$$\mathsf{AND-XOR}(\mathbf{ct}_1, \mathbf{ct_2}, \mathbf{ct}_3) = \left(\frac{1}{8}, \mathbf{0}\right) + 2 \cdot (\mathbf{ct}_1 + \mathbf{ct}_2) + 4 \cdot \mathbf{ct}_3$$

We now discuss the correctness of a 3-input AND-XOR gate without bootstrapping[6]. From the definition of the TFHE decryption procedure, the phase computed during decryption of a 3-input AND-XOR can be written as,

$$\Phi = \frac{1}{8} + 2 \cdot (\Delta(m_1) + \Delta(m_2)) + 4 \cdot \Delta(m_3) + 2 \cdot (e_1 + e_2) + 4 \cdot e_3$$

Here, m_i and e_i are the underlying plaintext bit and noise respectively in the input ciphertext $\mathbf{ct}_i$, $\forall i \in \{1, 2, 3\}$. Now, consider the analysis of the decryption operation with plaintext values $m_1, m_2, m_3 = 0$, i.e., $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = -\frac{1}{16}$.

$$\begin{aligned}
\Phi &= \frac{1}{8} + 2 \cdot (\Delta(m_1) + \Delta(m_2)) + 4 \cdot \Delta(m_3) + 2 \cdot (e_1 + e_2) + 4 \cdot e_3, \\
&= \frac{1}{8} + 2 \cdot \left(-\frac{1}{16} - \frac{1}{16}\right) + 4 \cdot \left(-\frac{1}{16}\right) + 2 \cdot (e_1 + e_2) + 4 \cdot e_3, \\
&= -\frac{3}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3), \\
&= -\frac{3}{8} + \delta \quad \left[|\delta| < 2^{-10}, \text{ as } |e_1|, |e_2|, |e_3| < 2^{-13}\right].
\end{aligned}$$

[6]Generally, a bootstrapping operation is a method of homomorphic decryption. Therefore, a Boolean gate correctness should hold even without a bootstrapping operation

In this cases, $\Phi \approx -\frac{3}{8}$, or, $\Phi \pmod 1 = -\frac{3}{8} < 0$, hence, the decryption of 3-input AND-XOR gate upon $m_1 = m_2 = m_3 = 0$ is 0, which is a correct decryption as $XOR(AND(0,0),0) = 0$. Now, consider when the plaintexts are $m_1 = m_2 = m_3 = 1$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = \frac{1}{16}$,

$$\Phi = \frac{5}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = \frac{5}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx \frac{5}{8}$, or, $\Phi \pmod 1 \approx -\frac{3}{8} < 0$ hence, the decryption of 3-input AND-XOR upon $m_1 = m_2 = m_3 = 1$ is 0, which is a correct decryption as $XOR(AND(1,1),1) = 0$. Now, consider when the first two inputs are encryptions of 0, i.e., $m_1 = m_2 = 0$, and consider $m_3 = 1$, or, $\Delta(m_1) = \Delta(m_2) = -\frac{1}{16}$, and $\Delta(m_3) = \frac{1}{16}$,

$$\Phi = \frac{1}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = \frac{1}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx \frac{1}{8}$, or, $\Phi \pmod 1 = \frac{1}{8} > 0$, hence the decryption of 3-input AND-XOR upon $m_1 = m_2 = 0$, and $m_3 = 1$ is 1, which is a correct decryption as $XOR(AND(0,0),1) = 1$. Now, consider when the first two inputs are encryptions of 1, i.e., $m_1 = m_2 = 1$, and consider $m_3 = 0$, or, $\Delta(m_1) = \Delta(m_2) = \frac{1}{16}$, and $\Delta(m_3) = -\frac{1}{16}$,

$$\Phi = \frac{1}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = \frac{1}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx \frac{1}{8}$, or, $\Phi \pmod 1 \approx \frac{1}{8} > 0$ hence, the decryption of 3-input AND-XOR upon $m_1 = m_2 = 1$, and $m_3 = 0$ is 1, which is a correct decryption as $XOR(AND(1,1),0) = 1$. Now, consider when the first two inputs are different, i.e., $m_1 = 0, m_2 = 1$, and consider $m_3 = 0$, or, $\Delta(m_1) = -\frac{1}{16}, \Delta(m_2) = \frac{1}{16}$, and $\Delta(m_3) = -\frac{1}{16}$,

$$\Phi = -\frac{1}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = -\frac{1}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx -\frac{1}{8}$, or, $\Phi \pmod 1 \approx -\frac{1}{8} < 0$ hence, the decryption of 3-input AND-XOR upon $m_1 = 0, m_2 = 1$, and $m_3 = 0$ is 0, which is a correct decryption as $XOR(AND(0,1),0) = 0$. Again, consider when the first two inputs are different, i.e., $m_1 = 0, m_2 = 1$, but $m_3 = 1$, or, $\Delta(m_1) = -\frac{1}{16}, \Delta(m_2) = \frac{1}{16}$, and $\Delta(m_3) = \frac{1}{16}$,

$$\Phi = \frac{3}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = \frac{3}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx \frac{3}{8}$, or, $\Phi \pmod 1 \approx \frac{3}{8} > 0$ hence, the decryption of 3-input AND-XOR upon $m_1 = 0, m_2 = 1$, and $m_3 = 1$ is 1, which is a correct decryption as $XOR(AND(0,1),1) = 1$. Therefore, with the above analysis, we observe that the combined AND-XOR gate gives the correct output for all possible input combinations. This finally proves the correctness of our proposed AND-XOR gate.

□

Lemma 2. *In TFHE, a binary OR gate followed by a binary XOR gate can be combined into an equivalent* 3*-input OR-XOR gate.*

Proof. We define the structure of the OR-XOR gate as follows,

$$\mathsf{OR}-\mathsf{XOR}(\mathbf{ct}_1,\mathbf{ct_2},\mathbf{ct}_3)=\left(\frac{3}{8},\mathbf{0}\right)+2\cdot(\mathbf{ct}_1+\mathbf{ct}_2)+4\cdot\mathbf{ct}_3$$

The phase computed during decryption of a 3-input OR-XOR can be written as,

$$\Phi=\frac{3}{8}+2\cdot(\Delta(m_1)+\Delta(m_2))+4\cdot\Delta(m_3)+2\cdot(e_1+e_2)+4\cdot e_3$$

.

Consider the analysis of the decryption operation of with $m_1, m_2, m_3 = 0$, i.e., $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = -\frac{1}{16}$.

$$\begin{aligned}\Phi&=\frac{3}{8}+2\cdot(\Delta(m_1)+\Delta(m_2))+4\cdot\Delta(m_3)+2\cdot(e_1+e_2)+4\cdot e_3,\\&=\frac{3}{8}+2\cdot\left(-\frac{1}{16}-\frac{1}{16}\right)+4\cdot\left(-\frac{1}{16}\right)+2\cdot(e_1+e_2)+4\cdot e_3,\\&=-\frac{1}{8}+2\cdot(e_1+e_2+2\cdot e_3),\\&=-\frac{1}{8}+\delta\quad\left[|\delta|<2^{-10},\text{ as }|e_1|,|e_2|,|e_3|<2^{-13}\right].\end{aligned}$$

In this cases, $\Phi \approx -\frac{1}{8}$, or, $\Phi \pmod 1 = -\frac{1}{8} < 0$, hence, the decryption of 3-input OR-XOR gate upon $m_1 = m_2 = m_3 = 0$ is 0, which is a correct decryption as $XOR(OR(0, 0), 0) = 0$. Now, consider when the plaintexts are $m_1 = m_2 = m_3 = 1$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = \frac{1}{16}$,

$$\Phi=\frac{7}{8}+2\cdot(e_1+e_2+2\cdot e_3)=\frac{7}{8}+\delta\quad\left[\text{as }|\delta|<2^{-10}\right].$$

In this cases, $\Phi \approx \frac{7}{8}$, or, $\Phi \pmod 1 \approx -\frac{1}{8} < 0$ hence, the decryption of 3-input OR-XOR upon $m_1 = m_2 = m_3 = 1$ is 0, which is a correct decryption as $XOR(OR(1, 1), 1) = 0$. Now, consider when the first two inputs are encryptions of 0, i.e., $m_1 = m_2 = 0$, and consider $m_3 = 1$, or, $\Delta(m_1) = \Delta(m_2) = -\frac{1}{16}$, and $\Delta(m_3) = \frac{1}{16}$,

$$\Phi=\frac{3}{8}+2\cdot(e_1+e_2+2\cdot e_3)=\frac{3}{8}+\delta\quad\left[\text{as }|\delta|<2^{-10}\right].$$

In this cases, $\Phi \approx \frac{3}{8}$, or, $\Phi \pmod 1 = \frac{3}{8} > 0$, hence the decryption of 3-input OR-XOR upon $m_1 = m_2 = 0$, and $m_3 = 1$ is 1, which is a correct decryption as $XOR(OR(0, 0), 1) = 1$. Now, consider when the first two inputs are encryptions of 1, i.e., $m_1 = m_2 = 1$, and consider $m_3 = 0$, or, $\Delta(m_1) = \Delta(m_2) = \frac{1}{16}$, and $\Delta(m_3) = -\frac{1}{16}$,

$$\Phi = \frac{3}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = \frac{3}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx \frac{3}{8}$, or, $\Phi \pmod 1 \approx \frac{3}{8} > 0$ hence, the decryption of 3-input OR-XOR upon $m_1 = m_2 = 1$, and $m_3 = 0$ is 1, which is a correct decryption as $XOR(OR(1,1),0) = 1$. Now, consider when the first two inputs are different, i.e., $m_1 = 0, m_2 = 1$, and consider $m_3 = 0$, or, $\Delta(m_1) = -\frac{1}{16}, \Delta(m_2) = \frac{1}{16}$, and $\Delta(m_3) = -\frac{1}{16}$,

$$\Phi = \frac{1}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = \frac{1}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx \frac{1}{8}$, or, $\Phi \pmod 1 \approx \frac{1}{8} > 0$ hence, the decryption of 3-input OR-XOR upon $m_1 = 0, m_2 = 1$, and $m_3 = 0$ is 1, which is a correct decryption as $XOR(OR(0,1),0) = 1$. Again, consider when the first two inputs are different, i.e., $m_1 = 0, m_2 = 1$, but $m_3 = 1$, or, $\Delta(m_1) = -\frac{1}{16}, \Delta(m_2) = \frac{1}{16}$, and $\Delta(m_3) = \frac{1}{16}$,

$$\Phi = \frac{5}{8} + 2 \cdot (e_1 + e_2 + 2 \cdot e_3) = \frac{5}{8} + \delta \quad \left[\text{as } |\delta| < 2^{-10}\right].$$

In this cases, $\Phi \approx \frac{5}{8}$, or, $\Phi \pmod 1 \approx -\frac{3}{8} < 0$ hence, the decryption of 3-input OR-XOR upon $m_1 = 0, m_2 = 1$, and $m_3 = 1$ is 0, which is a correct decryption as $XOR(OR(0,1),1) = 0$. Therefore, similarly using the above analysis, we observe that the combined OR-XOR gate gives the correct output for all possible input combinations. This proves the correctness of our proposed OR-XOR gate.

□

Lemma 3. *In TFHE, a binary NAND followed by a binary XOR can be combined into an equivalent* 3*-input NAND-XOR gate.*

Proof. We define the structure of the NAND-XOR gate in TFHE as follows,

$$\mathsf{NAND-XOR}(\mathbf{ct}_1, \mathbf{ct_2}, \mathbf{ct}_3) = \left(\frac{3}{8}, \mathbf{0}\right) - 2 \cdot (\mathbf{ct}_1 + \mathbf{ct}_2) + 4 \cdot \mathbf{ct}_3$$

The correctness of this composite gate can be similarly demonstrated by following the proof procedure for the AND-XOR gate, as the only differences are the constant terms.

□

Lemma 4. *In TFHE, a binary NOR gate followed by a binary XOR gate can be combined into an equivalent* 3*-input NOR-XOR gate.*

Proof. We define the structure of the NOR-XOR gate in TFHE as follows,

$$\mathsf{NOR-XOR}(\mathbf{ct}_1, \mathbf{ct_2}, \mathbf{ct}_3) = \left(\frac{1}{8}, \mathbf{0}\right) - 2 \cdot (\mathbf{ct}_1 + \mathbf{ct}_2) + 4 \cdot \mathbf{ct}_3$$

The correctness of this composite gate can be similarly demonstrated by following the proof procedure for the OR-XOR gate, as the only differences are the constant terms. □

By proving the above lemmas, we can conclusively assert that any binary gate followed by an XOR gate can be merged into an equivalent composite gate within the TFHE framework. Furthermore, Lemmas 1 through 4 can be collectively unified into the following theorem.

Theorem 2. *In TFHE, any binary gate followed by a binary XOR gate can be combined into an equivalent* 3*-input Boolean gate.*

3.2 4-input Homogeneous Gate Construction

Thus far, we have explored several new composite gate combinations, including AND-XOR, OR-XOR, etc. Now, we introduce two new multi-input Boolean gate structures, specifically 4-input Boolean AND and OR. Additionally, we discuss the challenges that have hindered the construction of 4-input versions of Boolean XOR operations.

Theorem 3. *In TFHE, consecutive* 3 *binary AND operations can be integrated into one* 4*-input Boolean AND operation.*

We define our 4-input AND gate in TFHE as follows,

$$\mathsf{AND}_4(\mathbf{ct}_1, \mathbf{ct}_2, \mathbf{ct}_3, \mathbf{ct}_4) = \left(-\frac{3}{16}, \mathbf{0}\right) + \mathbf{ct}_1 + \mathbf{ct}_2 + \mathbf{ct}_3 + \mathbf{ct}_4.$$

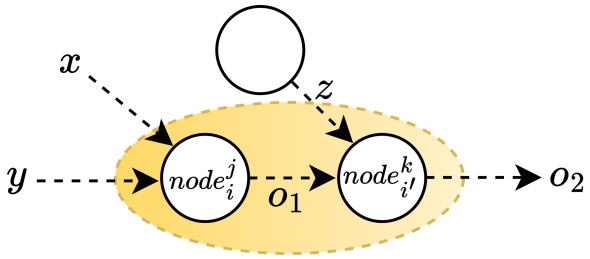
Theorem 4. *In TFHE, consecutive* 3 *binary OR operations can be integrated into one* 4*-input Boolean OR operation.*

We define our 4-input OR gate in TFHE as follows,

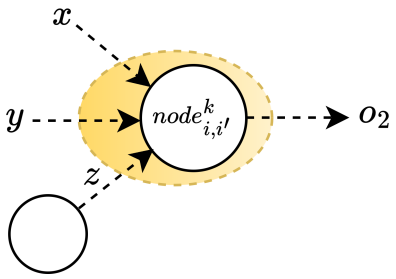
$$\mathsf{OR}_4(\mathbf{ct}_1, \mathbf{ct}_2, \mathbf{ct}_3, \mathbf{ct}_4) = \left(\frac{3}{16}, \mathbf{0}\right) + \mathbf{ct}_1 + \mathbf{ct}_2 + \mathbf{ct}_3 + \mathbf{ct}_4.$$

Theorem 5. *In TFHE, consecutive* 3 *binary XOR operations can not be integrated into one* 4*-input Boolean XOR operation, and therefore no such structure can be found for a* 4*-input XOR gate.*

The proofs of our 4-input Boolean gates are presented in Appendix A. Using these multi-input composite and homogeneous gates, we now demonstrate our circuit optimization procedure, which follows the blueprint of optimizing a Boolean circuit by optimizing its corresponding DAG representation. In the next section, we discuss our Boolean circuit synthesis technique via convex sub-graphs rewriting within the DAG representation of the input circuit. We show that our proposed technique can substantially reduce the size of a Boolean circuit in terms of the number of Boolean gates and, consequently, the number of bootstrapping operations.



(a) Convex sub-graph representing consecutive gates.



(b) Compact sub-graph structure.

Figure 3: Convex sub-graph optimization.

3.3 Our Proposed Circuit Synthesis Using Convex Sub-graph Optimization

In this section, we provide a detailed description of our proposed circuit synthesis framework using convex sub-graph optimization. Our framework takes a Boolean circuit $\mathcal{C}$ as input and converts it into an equivalent DAG representation $\mathcal{G}$. Every binary gate $gate_\ell^m$ in $\mathcal{C}$ is treated as a node $node_\ell^m$ in $\mathcal{G}$, and directed edges are established according to the connectivity of wires between gates in $\mathcal{C}$. For instance, if the output of a binary gate $gate_\ell^m$ at the m^{th} level is used as an input to another binary gate $gate_\ell^n$ at the n^{th} level, where $n > m$, then a directed edge is drawn from node $node_\ell^m$ to $node_\ell^n$ in $\mathcal{G}$. Here, $node_\ell^m$ and $node_\ell^n$ represent the corresponding nodes for the binary gates $gate_\ell^m$ and $gate_\ell^n$, respectively. Our circuit synthesis method involves parsing $\mathcal{C}$ to identify specific structures where the output of a binary gate $gate_i^j$ at the j^{th} level serves as an input to another binary gate $gate_{i'}^k$ at the k^{th} level, with $k > j$, and where the intermediate result or output of the first gate, $gate_i^j$, is not used elsewhere in the circuit, as depicted in Figure 2a. If, $i \neq i'$ it indicates that the two gates correspond to different binary operations, whereas if $i = i'$, it signifies that both gates perform the same binary operation. If our framework detects such a structure in $\mathcal{C}$ and $i = i'$, then it replaces the structure with a 3-input version of $gate_i$ at the k^{th} level. When $i \neq i'$, the structure is replaced with a possible 3-input composite gate $gate_{i,i'}$ at the k^{th} level with inputs x, y, and z, and the output be o_2 as depicted in Figure 2b. Section 3.1 provides a detailed illustration on the possible 3-input composite gate constructions. In order to utilize our 4-input homogeneous gates (illustrated in Section 3.2), our parser tries to find structures consisting of three gates, where every intermediate output is only used in the following gates.

Our circuit synthesizer map these consecutive gate structures in $\mathcal{C}$ to convex sub-graphs in $\mathcal{G}$. These sub-graphs are of size two or three because in TFHE we are only able to combine 2 or 3 binary gates into their equivalent 3 or 4-input Boolean gates. These sub-graphs should adhere to the convexity property, meaning there should be no outgoing edge from the sub-graph except for the output of the last node in topological order. Figure 3a shows the equivalent sub-graph representation of the consecutive gate structure presented in Figure 2a. The highlighted area in Figure 3a indicates the convex region of the sub-graph, where the only outgoing edge is o_2, which represents the output of the sub-graph computation. Consequently, our problem of circuit optimization reduces to the problem of convex sub-graph rewriting. In this optimization, we first identify such a convex sub-graph (as shown in Figure 3a) within a DAG representing our input circuit and replace the sub-graph with a more compact structure containing a multi-input node, which refers to an addition of a multi-input gate in the input circuit, as illustrated in Figure 3b. In the following paragraph, we provide a more detailed discussion on our circuit synthesis flow using convex sub-graph optimization using Algorithm 1.

Algorithm 1 DAG Optimization Algorithm

Require: A DAG $\mathcal{G}$ representing the input circuit $\mathcal{C}$.
Ensure: An optimized DAG $\mathcal{G}_{opt}$ representing the optimized circuit $\mathcal{C}_{opt}$.

```
Initialize a empty set Rmv_Set ← {}
for each node v_i in G do
    Initialize num_edges with fanout(v_i)
    if num_edges = 1 then
        Extract the target node v_j corresponding to e_{v_i,v_j}
        if level(v_j) = L and L ≠ X then
            if level(v_i) = L' and L = L' then
                Rmv_Set ← Rmv_Set ∪ {v_j}
                for each v_k ∈ G do
                    if ∃ e_{v_k,v_j} and v_k ≠ v_i then
                        Add e_{v_k,v_i} in G
                        Set level(v_i) = LL
                        for each child v_l of node v_j ∈ G do
                            Add e_{v_i,v_l}
        if level(v_i) == L and level(v_j) == X then
            Rmv_Set ← Rmv_Set ∪ {v_j}
            for each node v_k ∈ G do
                if ∃ e_{v_k,v_j} and v_k ≠ v_i then
                    Add e_{v_k,v_i} in G
                    Set level(v_i) = LX
                    for each child v_l of node v_j ∈ G do
                        Add e_{v_i,v_l}
Remove each node v_f ∈ Rmv_Set from G.
G_opt ← G
```

Our Circuit Synthesis Flow. Here, we present a detailed discussion of our proposed circuit synthesis flow, which enable us to design an automated framework that converts a Boolean circuit consisting of only binary gates into an efficient representation by introducing multi-input Boolean gates. Our proposed circuit synthesis flow has two steps– ① *Convert circuit into DAG* and ② *Optimizing convex sub-graphs in DAG*. In this work, we use Bristol format [bri([n. d.])] circuits due to its convenient representation and simpler processing overhead for FHE applications. We present a detailed discussion of the Bristol format circuits and the conversion algorithm into a DAG representation in Appendix D. Here, we present our circuit synthesis algorithm in a generalized manner, allowing it to be extended to any

circuit representation. Let us assume our input circuit is $\mathcal{C}$ that consists of only binary gates and our goal is to re-synthesize $\mathcal{C}$ to generate an efficient representation (say $\mathcal{C}_{opt}$) which may contain multi-input Boolean gates.

Convert $\mathcal{C}$ into DAG. Here, we discuss the conversion algorithm that takes $\mathcal{C}$ as input and convert it into an equivalent DAG representation $\mathcal{G}$. Our conversion algorithm simply parses the entire circuit $\mathcal{C}$ and creates a disconnected graph with a total number of nodes (say N) equal to the total binary gates present in $\mathcal{C}$. Now, for every binary operation $g \in \mathcal{C}$ identify the corresponding vertex/node $v_g \in \mathcal{G}$ and assign a label to v_g as $\mathcal{L}$, denoted as $level(v_g) = \mathcal{L}$. Here, $\mathcal{L} = \mathcal{A}$ if g is an AND operation; similarly, $\mathcal{L}$ can be $\mathcal{X}, \mathcal{O}, \mathcal{N}$ for g being XOR, OR and NOT respectively. For each input i_1 and i_2 of g identify the binary gates v_{g_1} and v_{g_2} whose outputs are i_1 and i_2 respectively. If such i_1 and i_2 are found then add two directed edges from v_{g_1} to v_g and v_{g_2} to v_g in $\mathcal{G}$.

Optimizing convex sub-graphs in $\mathcal{G}$. Given a DAG representation $\mathcal{G}$ of a Boolean circuit $\mathcal{C}$, our graph optimization algorithm begins by identifying convex sub-graphs within $\mathcal{G}$. A sub-graph of $\mathcal{G}$, containing two or three nodes, is considered convex if no directed edges exit the sub-graph except for the final output from the last node in the topological order (see Figure 3a). We identify such potential consecutive gate combinations in $\mathcal{C}$ by locating convex sub-graphs (of size two/three) within $\mathcal{G}$. The convexity property of these sub-graphs guarantees that the intermediate result is used only in the subsequent binary gate, as required for optimization. Our Algorithm 1 takes $\mathcal{G}$ of the target circuit $\mathcal{C}$ and first identifies the convex sug-graphs within $\mathcal{G}$ and replaces it with a compact structure that represents a $3, 4$-input Boolean gate. After optimizing all possible convex sub-graphs in $\mathcal{G}$, it converts into a corresponding Boolean circuit $\mathcal{C}_{opt}$, which is considered as efficient for TFHE.

We now discuss our graph optimization algorithm (described in Algorithm 1) in detail. In this algorithm, $fanout(v)$ refers to the total number of outgoing edges from node v, $e_{u,v}$ denotes a directed edge from node u to node v. This algorithm takes a DAG $\mathcal{G}$ representing a Boolean circuit $\mathcal{C}$ as input and returns an optimized DAG $\mathcal{G}_{opt}$ representing an efficient circuit $\mathcal{C}_{opt}$ for TFHE. Our DAG optimization algorithm starts by initializing an empty set called Rmv_Set, which stores the redundant nodes inside each convex sub-graph that need to be removed after replacing the sub-graph with a more compact structure. The algorithm then iterates over each node v_i in $\mathcal{G}$, and for each node, it calculates $fanout(v_i)$, stored in a variable called num_edges. If $num_edges = 1$, meaning the node v_i has only one outgoing edge, referring to a possible convex sub-graph starting with node v_i. Now, the algorithm extracts the target node v_j corresponding to this outgoing edge. Next, it checks whether $level(v_j)$ is $\mathcal{X}$. If $level(v_j) \neq \mathcal{X}$ i.e., the corresponding binary gate for node v_j is not an XOR operation, then it checks if $level(v_i)$ is equal to $level(v_j)$. If $level(v_j) = level(v_i)$, then node v_j is added to Rmv_Set, marking it for removal. The algorithm then examines each node $v_k \in \mathcal{G}$ and if there exists a directed edge e_{v_k,v_j} from v_k to v_j and $v_k \neq v_i$, a directed edge e_{v_k,v_i} is added from v_k to v_i. Node v_i is relabeled as $\mathcal{LL}$ to signify its new combined functionality, and any child v_l of v_j is connected to v_i, effectively bypassing v_j. This is how we insert 3-input Boolean gates in place of consecutive structures in $\mathcal{C}$ by reducing a convex sub-graph in $\mathcal{G}$. Similarly, if node v_j is labeled as $\mathcal{X}$ i.e., v_j corresponds to an XOR operation, the algorithm adds v_j to Rmv_Set and proceeds to check each node v_k with an edge to v_j, ensuring $v_k \neq v_i$. A directed edge e_{v_k,v_i} is then added and v_i is relabeled as $\mathcal{LX}$. Again, each child v_l of v_j is connected to v_i. Finally, after processing all nodes, the algorithm removes every node present in Rmv_Set from $\mathcal{G}$, and sets $\mathcal{G}_{opt}$ as $\mathcal{G}$, completing

Table 1: Circuit synthesis time (in secs) of our proposed optimization algorithm based on convex sub-graph rewriting.

Circuit	**Input size**	**Output size**	**Synthesis time (in secs)**
64-bit add	376	251	0.054
64-bit mul	13,675	9,797	45.33
64-bit div	29,926	17,456	206.82
AES-128	36,663	26,315	334.95
AES-256	50,666	36,225	655.54
SHA-256	135,073	88,713	4250.25

Table 2: Parameters used in our experimental setting

Parameter	**Value**
q (Ciphertext modulus)	2^{32}
n (Secret key dimension)	630
σ_{min} (Minimum standard deviation of encryption noise)	2^{-15}
σ_{max} (Maximum standard deviation of encryption noise)	2^{-6}

the optimization process by eliminating redundant nodes and edges, thereby optimizing the DAG representation of the circuit. We then transform $\mathcal{G}_{opt}$ into $\mathcal{C}_{opt}$, which is considered as an efficient circuit representation for TFHE and we then homomorphically evaluate $\mathcal{C}_{opt}$ on encrypted inputs using TFHE. In the next section, we evaluate these $\mathcal{C}_{opt}$ circuits for multiple benchmarks and illustrate the effectiveness of our proposed circuit synthesis algorithm by comparing with the state-of-the-art methods in this domain.

Complexity Analysis of Our Sub-graph Rewriting. Algorithm 1 describes the way to find a convex sub-graph in a larger graph (DAG) representing the input Boolean circuit. Our Algorithm has a polynomial ($O(G^3)$) asymptotic complexity, where G is the total number of binary gates present in the input Boolean circuit. This technique can be scaled to any sized graph represented as DAG because our method is to find convex sub-graphs of input size of 3 or 4 within a larger graph. The work by Reddington et al. [Reddington and Atasu(2011)] shows that an algorithm for enumerating all convex sub-graphs of a DAG makes no more than $O(|V(D)|N_{in} + N_{out} + 1)$ recursive calls, where $|V(D)|$ is the number of vertices in the DAG, and N_{in} and N_{out} are input and output constraints, respectively. Therefore, when N_{in} and N_{out} are constants, the algorithm is of polynomial complexity. In our case, we only aim to find sub-graphs whose N_{in} is either 3 or 4 (representing 3-input or 4-input gate) with $N_{out} = 1$. Therefore, in our case we can leverage the polynomial time algorithm from the work by Reddington et al. [Reddington and Atasu(2011)] for fixed values of N_{in} and N_{out}. Another important point to note is that, for every circuit, we need to just run our Algorithm 1 once in the offline/pre-processing phase. However, in Table 1 we provide our circuit synthesis time for m ultiple benchmark circuits with increasing input size.

Table 3: Comparison of our circuit optimization technique with Mono et al. [Mono et al.(2024)] on multiple Boolean circuits.

Circuit	Input size	FHE Transpiler [Gorantala et al.(2021)] Eval. time (in sec)	Mono et al. [Mono et al.(2024)] Num boots.	Mono et al. [Mono et al.(2024)] Eval. time (in sec)	Ours Num boots.	Ours Eval. time (in sec)	Efficiency gain
add3	12	1.69	**2**	0.65	8	**0.14**	4.64×
add4	17	2.32	**2**	0.74	11	**0.19**	3.89×
calc	886	106.00	**242**	72.30	632	**10.74**	6.73×
img-blur	317	40.10	**28**	10.50	219	**3.72**	2.82×
img-ricker	343	43.00	**39**	12.40	239	**4.06**	3.05×
img-sharp	180	23.10	**31**	8.68	135	**2.3**	3.77×
structs3d	1100	138.00	**77**	32.90	739	**12.56**	2.62×
sum3d	958	120.00	**74**	29.10	709	**12.05**	2.41×
strrev	538	82.70	**266**	70.80	268	**4.6**	15.39×
ReLU	30	4.17	**15**	3.67	23	**0.39**	9.41×
sqrt	234	24.00	**64**	18.60	129	**2.19**	8.49×

Table 4: Comparison of our circuit optimization technique with Guan et al. [Guan et al.(2024)] and Yu et al. [Yu et al.(2023)] on EPFL benchmark suite.

Benchmark (EPFL)	Initial (# Gates)	Guan et al. [Guan et al.(2024)] (#Gates)	Yu et al. [Yu et al.(2023)] (#Gates)	Yu et al. [Yu et al.(2023)] Enhanced TechMap (#Gates)	Yu et al. [Yu et al.(2023)] Enhanced TechMap Eval. time (ms)	Ours (#Gates)	Ours Eval. time (ms)	Efficiency gain
Adder	890	507	191	**128**	4,864	572	7,436	—
Barrel shifter	3,141	2,496	2,496	2,496	99,840	**2,243**	**38,131**	2.6×
Divider	38,010	28,773	13,116	**13,076**	52,3040	25,402	**431,834**	1.2×
Log2	24,670	20,160	14,058	**13,573**	542,920	16,178	**275,026**	1.9×
Max	2,865	2,154	2,068	2,066	82,640	**1,793**	**30,481**	2.7×
Multiplier	19,416	14,530	10,442	**9,957**	398,280	13,186	**224,162**	1.8×
Sine	4,461	3,578	2,502	**2,398**	10,5512	3,004	**51,068**	2.1×
Square-root	19,203	10,452	8,276	**8,218**	328,720	12,604	**214,268**	1.5×
Square	14,584	11,968	8,661	**7,547**	301,880	10,467	**177,939**	1.7×
Round-robin arbiter	11,839	11,434	11605	11,605	464,200	**6,186**	**105,162**	4.4×
Coding-cavlc	685	554	545	542	21,680	**438**	**7,446**	2.9×
ALU control unit	124	97	94	94	3,760	**82**	**1,394**	2.7×
Decoder	304	**291**	292	292	11,680	304	**5,168**	2.2×
I2c controller	1,301	1,109	1,032	1,029	41,160	**840**	**14,280**	2.9×
Int-to-float converter	237	175	171	170	6,800	**149**	**2,533**	2.7×
Memory controller	46,442	36,446	35,510	35,016	1,540,704	**27,532**	**468,044**	3.3×
Priority encoder	974	833	818	818	32,720	**681**	**11,577**	2.8×
Look-ahead XY router	218	174	134	**126**	5,040	136	**2,312**	2.2×
Voter	8,282	5,170	3,306	**2,936**	117,440	5,991	**101,847**	1.2×

4 Experimental Results

This section outlines our experimental setup and presents results for multiple Boolean circuits using our proposed circuit synthesis framework. We also compare our approach with some recent state-of-the-art Boolean circuit optimization techniques. FHEW [Ducas and Micciancio(2015)] and TFHE [Chillotti et al.(2020a)] are two widely used FHE schemes for Boolean circuits. The primary contribution of our approach is to design multi-input gates with only one bootstrapping and the bootstrapping cost is same for both multi-input and binary gates. In contrast to the other works [Mono et al.(2024), Guan et al.(2024), Yu et al.(2023)] in which they propose multi-input gates by using multi-bit LUTs whose FBS cost grows super linearly with the number of inputs [Carpov(2024)] (in Appendix E we provide an argument supporting this). Another recent work by Singh et al. [Singh et al.(2023)] uses gate-level optimization in Boolean circuits for TFHE. They define new multi-input gates with one bootstrapping but their multi-input gates are limited to homogeneous gates with only three inputs, i.e., 3-input AND, XOR, etc. Whereas, our optimization techniques use composite gates such as 3-input AND-XOR, 3-input OR-XOR and additionally, we also introduce 4-input AND and 4-input OR gates. Due to this, the reduction in bootstrapping count using our approach also outperforms the work by Singh et al. [Singh et al.(2023)].

In the subsequent paragraphs we present detailed discussions on the comparison results for these state-of-the-art schemes on Boolean circuit optimization for FHEW and TFHE. We use the TFHE built-in functions for different binary gate evaluations implemented in

Table 5: Comparison of our circuit optimization technique with FHEDA [Singh et al.(2023)] on multiple Boolean circuit from SCALE-MAMBA benchmark suite. Here, "gate count" refers to the number of Boolean gates in the circuit.

Circuit	**Input circuit gate count**	**FHEDA [Singh et al.(2023)] (reduced gate count)**	**Our optimization (reduced gate count)**	**% Overall reduction in gate count**	**% Efficiency (w.r.t FHEDA)**
Arithmetic Functions					
64-bit add	376	312	**251**	33.24%	19.55%
64-bit sub	439	375	**314**	28.47%	16.27%
64-bit mul	13,675	11,722	**9,830**	28.12%	16.14%
64-bit div	29,926	21,984	**17,581**	41.25%	20.02%
64-bit negation	190	190	**189**	0.52%	0.52%
Mod add 512	9,720	9,720	**9,210**	5.25%	5.25%
Floating-point Operations					
FP-add	15,637	14,897	**11,787**	24.62%	20.87%
FP-mul	44,899	39,174	**33,821**	24.67%	13.67%
FP-lt	1,536	1,527	**1,463**	4.75%	4.2%
FP-f2i	3,932	3,880	**3,107**	20.98%	19.92%
FP-i2f	7,136	6,899	**5,263**	26.24%	23.71%
floor	1,613	1,507	**1,245**	22.81%	17.38%
ceil	1,618	1,513	**1,251**	22.68%	17.32%
Cryptographic Functions					
AES-128	36,663	30,614	**29,014**	20.86%	5.22%
AES-192	41,565	34,665	**32,873**	20.92%	5.16%
AES-256	50,666	42,246	**40,038**	20.98%	5.23%
SHA-256	135,073	111,625	**90,153**	33.25%	19.23%
Keccak	192,086	176,726	**138,326**	27.98%	21.73%
SHA-512	349,617	288,809	**232,249**	33.57%	24.35%

the original TFHE library[7] itself. Therefore, the parameters used in our experiments are identical to those specified in the TFHE library. We have explicitly listed certain parameters in Table 2. These parameters are estimated to achieve 127-bit security by the LWE estimator [Albrecht et al.(2015)]. All the experiments and our evaluation algorithms are implemented on a high-end workstation with an Intel(R) Xeon(R) CPU E5-2690 v4 CPU (2.60GHz clock-frequency), 28 physical cores, and 128GB RAM. Source code of our circuit synthesis algorithm is provided in: https://github.com/Animesh005/circuit_syn.git

Comparison with Mono et al. [Mono et al.(2024)]. Mono et al. [Mono et al.(2024)] proposed circuit optimization specific to FHEW-based schemes. Their approach consists of utilizing multi-bit LUTs and FAs followed by an efficient LUT grouping strategy and multi-value FBS. However, it requires a larger parameter set to support multi-bit LUT and multi-value FBS, significantly increasing the bootstrapping cost. Consequently, their circuit evaluation time increases although having fewer bootstrapping operations. Whereas, our approach focuses more on gate-level optimization without altering the TFHE parameter set. This allows us to achieve around 17 ms per gate evaluation (including $2, 3$, and 4-input gates). However, Mono et al. [Mono et al.(2024)] uses the parameter set tfhe-11-multi (mentioned in Table 4 in [Mono et al.(2024)]) which provides bootstrapping cost as 0.29 $secs$. Therefore, we achieve an average of $4 - 6\times$ of improvement in the homomorphic evaluation over encrypted input and this improvement can go up to $15\times$ depending on the circuits. Table 3 demonstrates the efficiency of our proposed technique compared to Mono et al. [Mono et al.(2024)] using the circuits presented in their work. Table 3 also shows a comparison with FHE transpiler [Gorantala et al.(2021)], which is a publicly available solution to circuit synthesis. These results are also taken from [Mono et al.(2024)]. Note that, FHEW-based schemes inherently exhibit slower bootstrapping (0.69 secs) compared to FHE-based schemes (17 ms), consequently, the multi-value FBS cost in FHEW is much higher than the multi-value FBS in TFHE. Therefore, in the next paragraph, we compare our optimization

[7]https://github.com/tfhe/tfhe

with schemes based on multi-value FBS in TFHE [Guan et al.(2024), Yu et al.(2023)].

Comparison with Guan et al. [Guan et al.(2024)] and Yu et al. [Yu et al.(2023)]. These works provide the state-of-the-art design of efficient circuits for TFHE by utilizing multi-input gates as LUTs followed by multi-value bootstrapping-based approach. Their implementation is built on the Concrete library, which utilizes a parameter set similar to that of TFHE, as shown in Table 2. Particularly, [Guan et al.(2024)] performs a post-processing after the technology mapping phase to merge multiple binary gates into multi-input gates, whereas [Yu et al.(2023)] utilizes multi-input gates in the technology mapping phase itself. Both of the works are then followed by a multi-value bootstrapping technique that allows to evaluate different binary gates having the same support (or, input) using one FBS cost [Carpov et al.(2019)]. [Yu et al.(2023)] additionally, proposes a NOT-gate reduction technique to increase the possibility of multi-value bootstrapping. Therefore, [Yu et al.(2023)] enhances the approaches proposed in [Guan et al.(2024)]. In Table 4 we compare our results with both [Guan et al.(2024)] and [Yu et al.(2023)].

The circuit synthesis approach in [Yu et al.(2023)] converts a Boolean circuit into an LUT network, where the Boolean gates are represented using LUTs and uses a multi-value bootstrapping technique to evaluate multiple LUTs with a single FBS. This method allows to combine multiple parallel binary gates into one composite gate. Here, we use parallel binary gates to refer different binary gates having same input support. In contrast, our circuit synthesis technique based on convex sub-graph optimization focuses on combining sequential binary gates, i.e., consecutive binary gates in which the output of one gate is only used as input to the following gate. This technique is able to outperform [Yu et al.(2023)] on several circuits from the EPFL benchmark suite in terms of the total number of bootstrapping operations. The reason is Yu et al. [Yu et al.(2023)] synthesizes Boolean circuits using multi-bit LUTs (ranging from 3-bit to 6-bit) and this method performs better for circuits like "Adder", "Divider" etc, which has a significant number of parallel gates that are merged into composite gates using multi-value FBS, and this technique is only possible in LUT-based gate definitions. LUT-based designs are particularly effective in representing sub-circuits as lookup tables, offering an efficient implementation for narrow (low-fanin) and deep sub-circuits. However, they become less efficient for wide (high-fanin) and shallow sub-circuits. In such cases, the cost of an FBS increases with the fan-in size, and due to the shallowness, the sub-circuits contain fewer binary gates. As a result, LUT-based designs may surpass the total gate bootstrapping cost of the un-optimized sub-circuits [Carpov(2024)]. On the other hand, for circuits like "Max", "Round-robin arbiter" etc which have narrow but shallow sub-circuits and having less number of parallel gates, LUT-based optimizations primarily rely on a higher number of 2-bit LUTs rather than 3-bit or 4-bit LUTs, with minimal LUT grouping. In such scenarios, our synthesis technique outperforms [Guan et al.(2024), Yu et al.(2023)] by merging sequential binary gates into multi-input gates. As a result, our approach not only provides better evaluation cost but also provides a lesser number of bootstrappings.

It is important to note that the cost of FBS is significantly higher than a gate-bootstrapping cost in TFHE and FBS cost grows super-linearly with the input size [Yu et al.(2023)]. Hence, the number of bootstrappings may continue to decrease with FBS input size, but the evaluation cost of the circuit consistently increases. This is because the reduction in the bootstrapping count is no longer sufficient to compensate for the increase of an FBS cost [Carpov(2024)]. Table 4 presents the evaluation cost for each circuit, where our binary and multi-input gates evaluate in just 17 ms –substantially lower than multi-input FBS.

Consequently, our approach achieves up to a 4× improvement in homomorphic evaluation cost compared to [Yu et al.(2023)]. The primary advantage of our approach is that we propose new multi-input gates with one gate-bootstrapping cost which is similar to a binary gate (17 ms); whereas [Guan et al.(2024), Yu et al.(2023)] proposes multi-input gate by using multi-bit LUT whose FBS cost grows super-linearly with the LUT input size, although having one bootstrapping.

Additionally, our circuit synthesis technique is more generic and efficient, optimizing at the gate level to reduce area and enhance ASIC compatibility. In contrast, LUT-based representations suit FPGA designs but are difficult to implement on ASICs, as LUTs must first be mapped to Boolean structures using only binary gates. Furthermore, LUT-based circuits have higher space complexity, as they store encrypted bits. An n-bit LUT requires approximately $2^n \times 2$KB (per ciphertext size in TFHE), whereas a gate-level representation needs only 4 to 5 bits to specify gate types.

Thus, LUT-based optimizations are advantageous for a limited set of circuits having some specific sub-structures. Therefore, we also compare our results with the state-of-the-art gate-based circuit optimization method for TFHE proposed by Singh et al. [Singh et al.(2023)].

Comparison with Singh et al. [Singh et al.(2023)]. FHEDA [Singh et al.(2023)] utilizes conventional circuit synthesis tools such as Cadence Genus and the standard cell library TSL18FS120 from Tower Semiconductor Ltd., at the 180nm technology node. They augmented these cell libraries to include definitions for only 3-input homogeneous gates, whereas our method uses composite gates alongwith homogeneous gates supporting up to 4-inputs. Note that, the multi-input gates in both FHEDA and our work have the same evaluation cost of 17 ms. However, we outperform FHEDA because of our efficient circuit synthesis technique and more number of multi-input gates. During circuit synthesis in FHEDA, the Cadence tool re-writes a circuit's functionality using both 2 and 3-input gates, reducing the circuit size and thus decreasing the homomorphic evaluation time. In particular, FHEDA proposes four different cell libraries and proposes that an efficient cell library should consist of both $2, 3$-input AND, XOR gates, with NOT gates, present in the library set Lib_Set3 in [Singh et al.(2023)]. Therefore, in Table 5, we compare our approach with the optimized circuit produced by FHEDA using their Lib_Set3. We showcase three different classes of circuits from the benchmark suite developed by the authors of SCALE-MAMBA [Hastings et al.(2019)], referred to as Bristol format circuits (refer to Appendix B). These classes include– Arithmetic Functions, Floating-point Operations, and Cryptographic Functions. For most circuits in each class, we've significantly improved efficiency by reducing circuit size, leading to fewer bootstrapping operations. In particular, for the basic arithmetic circuits, we achieve around $16 - 20\%$ of more reduction and for other cryptographic objects like hash functions (SHA-256, Keccak, etc.) we are able to achieve around 20% of further reduction compared to FHEDA. Similarly, for the Bristol formats of floating point operations, we gain up to 20% reduction in circuit size.

5 Conclusion & Future Direction

In this work, we have presented an efficient circuit synthesis framework that utilizes multi-input composite and homogeneous gates and can significantly reduce the size of a Boolean circuit, thus resulting in fewer bootstrapping operations while evaluating under TFHE. Our

circuit synthesis framework converts a Boolean circuit into an equivalent DAG representation and rewrites the convex sub-graphs within DAG to realize a multi-input gate in the input circuit. Our proposed approach achieves significant improvement over the state-of-the-art Boolean circuit optimization frameworks that use LUT-based designs. These advancements highlight the potential of our approach to improve FHE computations, making them more feasible for practical use. As a future direction, a hybrid synthesis technique combining LUT-based and gate-based designs can be studied to utilize the benefits of both techniques. We also plan to integrate our circuit synthesis with ASIC design for TFHE.

References

[bri([n. d.])] [n. d.]. KU Leuven COSIC. SCALE-MAMBA, 2019. https://github.com/KULeuven-COSIC/SCALE-MAMBA.

[Albrecht et al.(2015)] Martin R Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology* 9, 3 (2015), 169–203.

[Bourse et al.(2020)] Florian Bourse, Olivier Sanders, and Jacques Traoré. 2020. Improved secure integer comparison via homomorphic encryption. In *Cryptographers' Track at the RSA Conference*. Springer, 391–416.

[Brakerski(2012)] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*. Springer, 868–886.

[Carpov(2024)] Sergiu Carpov. 2024. A fast heuristic for mapping Boolean circuits to functional bootstrapping. *Cryptology ePrint Archive* (2024).

[Carpov et al.(2019)] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New techniques for multi-value input homomorphic evaluation and applications. In *Cryptographers' Track at the RSA Conference*. Springer, 106–126.

[Chillotti et al.(2017)] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2017. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 377–408.

[Chillotti et al.(2020a)] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020a. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* 33, 1 (2020), 34–91.

[Chillotti et al.(2020b)] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2020b. Concrete: Concrete operates on ciphertexts rapidly by extending tfhe. In *WAHC 2020-8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*.

[Demmler et al.(2015)] Daniel Demmler, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. 2015. Automated synthesis of optimized circuits for secure computation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1504–1517.

[Ducas and Micciancio(2015)] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 617–640.

[Gentry(2009)] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[Ghodsi et al.(2020)] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. 2020. Cryptonas: Private inference on a relu budget. *Advances in Neural Information Processing Systems* 33 (2020), 16961–16971.

[Gorantala et al.(2021)] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, et al. 2021. A general purpose transpiler for fully homomorphic encryption. *arXiv preprint arXiv:2106.07893* (2021).

[Gouert et al.(2023a)] Charles Gouert, Vinu Joseph, Steven Dalton, Cedric Augonnet, Michael Garland, and Nektarios Georgios Tsoutsos. 2023a. Accelerated Encrypted Execution of General-Purpose Applications. *IACR Cryptol. ePrint Arch.* 2023 (2023), 641.

[Gouert et al.(2023b)] Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. 2023b. Helm: Navigating homomorphic encryption through gates and lookup tables. *Cryptology ePrint Archive* (2023).

[Guan et al.(2024)] Zhenyu Guan, Ran Mao, Qianyun Zhang, Zhou Zhang, Zian Zhao, and Song Bian. 2024. Autohog: Automating homomorphic gate design for large-scale logic circuit evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 7 (2024), 1971–1983.

[Hastings et al.(2019)] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. 2019. Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 1220–1237.

[Hu et al.(2024)] Xiao Hu, Zhihao Li, Zhongfeng Wang, and Xianhui Lu. 2024. ALT: Area-Efficient and Low-Latency FPGA Design for Torus Fully Homomorphic Encryption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2024).

[Jiang et al.(2022)] Lei Jiang, Qian Lou, and Nrushad Joshi. 2022. Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 235–240.

[Lyubashevsky et al.(2010)] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 1–23.

[Matsuoka et al.(2021a)] Kotaro Matsuoka, Ryotaro Banno, Naoki Matsumoto, Takashi Sato, and Song Bian. 2021a. Virtual secure platform: A {Five-Stage} pipeline processor over {TFHE}. In *30th USENIX security symposium (USENIX Security 21)*. 4007–4024.

[Matsuoka et al.(2021b)] Kotaro Matsuoka, Yusuke Hoshizuki, Takashi Sato, and Song Bian. 2021b. Towards better standard cell library: Optimizing compound logic gates for tfhe. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 63–68.

[Mono et al.(2024)] Johannes Mono, Kamil Kluczniak, and Tim Güneysu. 2024. Improved Circuit Synthesis with Multi-Value Bootstrapping for FHEW-like Schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024, 4 (2024), 633–656.

[Reddington and Atasu(2011)] Joseph Reddington and Kubilay Atasu. 2011. Complexity of computing convex subgraphs in custom instruction synthesis. *IEEE transactions on very large scale integration (VLSI) systems* 20, 12 (2011), 2337–2341.

[Regev(2009)] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 1–40.

[Singh et al.(2023)] Animesh Singh, Smita Das, Anirban Chakraborty, Rajat Sadhukhan, Ayantika Chatterjee, and Debdeep Mukhopadhyay. 2023. FHEDA: Efficient Circuit Synthesis with Reduced Bootstrapping for Torus FHE. *Cryptology ePrint Archive* (2023).

[Songhori et al.(2015)] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. 2015. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 411–428.

[Stracovsky et al.(2022)] Roy Stracovsky, Rasoul Akhavan Mahdavi, and Florian Kerschbaum. 2022. Faster evaluation of AES using TFHE. *Poster Session, FHE. Org* (2022).

[Trama et al.(2023)] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. 2023. A homomorphic AES evaluation in less than 30 seconds by means of TFHE. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 79–90.

[Turan et al.(2020)] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. 2020. HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA. *IEEE Trans. Comput.* 69, 8 (2020), 1185–1196.

[Van Beirendonck et al.(2023)] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Furkan Turan, and Ingrid Verbauwhede. 2023. FPT: A fixed-point accelerator for torus fully homomorphic encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 741–755.

[Wang et al.(2024)] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. 2024. Circuit bootstrapping: faster and smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 342–372.

[Wei et al.(2023)] Benqiang Wei, Ruida Wang, Zhihao Li, Qinju Liu, and Xianhui Lu. 2023. Fregata: Faster homomorphic evaluation of aes via tfhe. In *International Conference on Information Security*. Springer, 392–412.

[Wolf(2016)] Clifford Wolf. 2016. Yosys open synthesis suite. (2016).

[Yu et al.(2023)] Mingfei Yu, Sergiu Carpov, Alessandro Tempia Calvino, and Giovanni De Micheli. 2023. On the Synthesis of High-performance Homomorphic Boolean Circuits. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 51–63.

A Proof of 4-input Homogeneous Gate Construction

Here, we will discuss the proof of Theorem 4, 5, and 6 that is a 4-input AND can be constructed (Theorem 4) a 4-input OR can be constructed (Theorem 5) but a 4-input XOR gate can not be constructed (Theorem 6) within the TFHE framework.

Proof of Theorem 4.

Theorem 4: In TFHE, consecutive 3 binary AND operations can be integrated into one 4-input Boolean AND operation.

Proof. We define our novel 4-input AND gate in the TFHE framework as follows,

$$\mathsf{AND}_4(\mathbf{ct}_1, \mathbf{ct}_2, \mathbf{ct}_3, \mathbf{ct}_4) = \left(-\frac{3}{16}, \mathbf{0}\right) + \mathbf{ct}_1 + \mathbf{ct}_2 + \mathbf{ct}_3 + \mathbf{ct}_4.$$

We now prove the correctness of this Boolean gate similarly by analyzing each possible input combination. From the definition of the TFHE decryption function, the phase computed during decryption of after a 4-input AND can be written as,

$$\Phi = -\frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4)$$

.

Consider the analysis of the decryption operation when all the underlying plaintext values are 0, i.e., $m_1, m_2, m_3, m_4 = 0$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = \Delta(m_4) = -\frac{1}{16}$.

$$\begin{aligned}
\Phi &= -\frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4), \\
&= -\frac{3}{16} + \left(-\frac{1}{16} - \frac{1}{16} - \frac{1}{16} - \frac{1}{16}\right) + (e_1 + e_2 + e_3 + e_4), \\
&= -\frac{7}{16} + (e_1 + e_2 + e_3 + e_4), \\
&= -\frac{7}{16} + \delta \quad \left[|\delta| < 2^{-11}, \text{ as } |e_1|, |e_2|, |e_3|, |e_4| < 2^{-13}\right].
\end{aligned}$$

In this cases, $\Phi \approx -\frac{7}{16}$, or, $\Phi \pmod 1 = -\frac{7}{16} < 0$, hence, the decryption of 4-input AND gate upon $m_1 = m_2 = m_3 = m_4 = 0$ is 0, which is a correct decryption. Now, consider when one of the plaintexts is 0, i.e., without loss of generality consider, $m_1 = 0$ and $m_2 = m_3 = m_4 = 1$, or, $\Delta(m_1) = -\frac{1}{16}$ and $\Delta(m_2) = \Delta(m_3) = \Delta(m_4) = \frac{1}{16}$.

$$\Phi = -\frac{1}{16} + (e_1 + e_2 + e_3 + e_4) = -\frac{1}{16} + \delta \quad \left[\text{as } |\delta| < 2^{-11}\right].$$

In this cases, $\Phi \approx -\frac{1}{16}$, or, $\Phi \pmod 1 = -\frac{1}{16} < 0$, hence, the decryption of 4-input AND gate upon $m_1 = 0$ and $m_2 = m_3 = m_4 = 1$ is 0, which is a correct decryption. Now, consider when two of the plaintexts are 0, i.e., without loss of generality consider, $m_1 = m_2 = 0$ and $m_3 = m_4 = 1$, or, $\Delta(m_1) = \Delta(m_2) = -\frac{1}{16}$ and $\Delta(m_3) = \Delta(m_4) = \frac{1}{16}$.

$$\Phi = -\frac{3}{16} + (e_1 + e_2 + e_3 + e_4) = -\frac{3}{16} + \delta \quad \left[\text{ as } |\delta| < 2^{-11}\right].$$

In this cases, $\Phi \approx -\frac{3}{16}$, or, $\Phi \pmod 1 = -\frac{3}{16} < 0$, hence, the decryption of 4-input AND gate upon $m_1 = m_2 = 0$ and $m_3 = m_4 = 1$ is 0, which is a correct decryption. Now, consider when three of the plaintexts are 0, i.e., without loss of generality consider, $m_1 = m_2 = m_3 = 0$ and $m_4 = 1$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = -\frac{1}{16}$ and $\Delta(m_4) = \frac{1}{16}$.

$$\Phi = -\frac{5}{16} + (e_1 + e_2 + e_3 + e_4) = -\frac{5}{16} + \delta \quad \left[\text{as } |\delta| < 2^{-11}\right].$$

In this cases, $\Phi \approx -\frac{5}{16}$, or, $\Phi \pmod 1 = -\frac{5}{16} < 0$, hence, the decryption of 4-input AND gate upon $m_1 = m_2 = m_3 = 0$ and $m_4 = 1$ is 0, which is a correct decryption. Now, consider the final combination when all of the plaintexts are 1, i.e, $m_1 = m_2 = m_3 = m_4 = 1$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = \Delta(m_4) = \frac{1}{16}$.

$$\Phi = \frac{1}{16} + (e_1 + e_2 + e_3 + e_4) = \frac{1}{16} + \delta \quad \left[\text{ as } |\delta| < 2^{-11}\right].$$

In this cases, $\Phi \approx \frac{1}{16}$, or, $\Phi \pmod 1 = \frac{1}{16} > 0$, hence, the decryption of 4-input AND gate upon $m_1 = m_2 = m_3 = m_4 = 1$ is 1, which is a correct decryption. Hence, following the above analysis, we observe that our proposed 4-input AND gate gives the correct output for all possible input combinations, proving this Boolean gate's correctness. □

Proof of Theorem 5.

Theorem 5: In TFHE, consecutive 3 binary OR operations can be integrated into one 4-input Boolean OR operation.

Proof. We define our novel 4-input OR gate in the TFHE framework as follows,

$$\mathsf{OR}_4(\mathbf{ct}_1, \mathbf{ct}_2, \mathbf{ct}_3, \mathbf{ct}_4) = \left(\frac{3}{16}, \mathbf{0}\right) + \mathbf{ct}_1 + \mathbf{ct}_2 + \mathbf{ct}_3 + \mathbf{ct}_4.$$

We now prove the correctness of this Boolean gate similarly by analyzing each possible input combination. From the definition of the TFHE decryption function, the phase computed during decryption of a 4-input OR can be written as,

$$\Phi = \frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4)$$

.

Consider the analysis of the decryption operation when all the underlying plaintext values are 0, i.e., $m_1, m_2, m_3, m_4 = 0$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = \Delta(m_4) = -\frac{1}{16}$.

$$\begin{aligned}\Phi &= \frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{3}{16} + \left(-\frac{1}{16} - \frac{1}{16} - \frac{1}{16} - \frac{1}{16}\right) + (e_1 + e_2 + e_3 + e_4),\\ &= -\frac{1}{16} + (e_1 + e_2 + e_3 + e_4),\\ &= -\frac{1}{16} + \delta \quad \left[|\delta| < 2^{-11},\ \text{as } |e_1|, |e_2|, |e_3|, |e_4| < 2^{-13}\right].\end{aligned}$$

In this cases, $\Phi \approx -\frac{1}{16}$, or, $\Phi \pmod 1 = -\frac{1}{16} < 0$, hence, the decryption of 4-input OR gate upon $m_1 = m_2 = m_3 = m_4 = 0$ is 0, which is a correct decryption. Now, consider when one of the plaintexts is 0, i.e., without loss of generality consider, $m_1 = 0$ and $m_2 = m_3 = m_4 = 1$, or, $\Delta(m_1) = -\frac{1}{16}$ and $\Delta(m_2) = \Delta(m_3) = \Delta(m_4) = \frac{1}{16}$.

$$\begin{aligned}\Phi &= \frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{3}{16} + \left(-\frac{1}{16} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16}\right) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{5}{16} + (e_1 + e_2 + e_3 + e_4) = \frac{5}{16} + \delta \quad \left[\text{as } |\delta| < 2^{-11}\right].\end{aligned}$$

In this cases, $\Phi \approx \frac{5}{16}$, or, $\Phi \pmod 1 = \frac{5}{16} > 0$, hence, the decryption of 4-input OR gate upon $m_1 = 0$ and $m_2 = m_3 = m_4 = 1$ is 1, which is a correct decryption. Now, consider when two of the plaintexts are 0, i.e., without loss of generality consider, $m_1 = m_2 = 0$ and $m_3 = m_4 = 1$, or, $\Delta(m_1) = \Delta(m_2) = -\frac{1}{16}$ and $\Delta(m_3) = \Delta(m_4) = \frac{1}{16}$.

$$\begin{aligned}\Phi &= \frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{3}{16} + \left(-\frac{1}{16} - \frac{1}{16} + \frac{1}{16} + \frac{1}{16}\right) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{3}{16} + (e_1 + e_2 + e_3 + e_4) = \frac{3}{16} + \delta \quad \left[\text{ as } |\delta| < 2^{-11}\right].\end{aligned}$$

In this cases, $\Phi \approx \frac{3}{16}$, or, $\Phi \pmod 1 = \frac{3}{16} > 0$, hence, the decryption of 4-input OR gate upon $m_1 = m_2 = 0$ and $m_3 = m_4 = 1$ is 1, which is a correct decryption. Now, consider when three of the plaintexts are 0, i.e., without loss of generality consider, $m_1 = m_2 = m_3 = 0$ and $m_4 = 1$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = -\frac{1}{16}$ and $\Delta(m_4) = \frac{1}{16}$.

$$\begin{aligned}\Phi &= \frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{3}{16} + \left(-\frac{1}{16} - \frac{1}{16} - \frac{1}{16} + \frac{1}{16}\right) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{1}{16} + (e_1 + e_2 + e_3 + e_4) = \frac{1}{16} + \delta \quad \left[\text{as } |\delta| < 2^{-11}\right].\end{aligned}$$

In this cases, $\Phi \approx \frac{1}{16}$, or, $\Phi \pmod 1 = \frac{1}{16} > 0$, hence, the decryption of 4-input OR gate upon $m_1 = m_2 = m_3 = 0$ and $m_4 = 1$ is 1, which is a correct decryption. Now, consider the final combination when all of the plaintexts are 1, i.e, $m_1 = m_2 = m_3 = m_4 = 1$, or, $\Delta(m_1) = \Delta(m_2) = \Delta(m_3) = \Delta(m_4) = \frac{1}{16}$.

$$\begin{aligned}\Phi &= \frac{3}{16} + (\Delta(m_1) + \Delta(m_2) + \Delta(m_3) + \Delta(m_4)) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{3}{16} + \left(\frac{1}{16} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16}\right) + (e_1 + e_2 + e_3 + e_4),\\ &= \frac{7}{16} + (e_1 + e_2 + e_3 + e_4) = \frac{7}{16} + \delta \quad \left[\text{ as } |\delta| < 2^{-11}\right].\end{aligned}$$

In this cases, $\Phi \approx \frac{7}{16}$, or, $\Phi \pmod 1 = \frac{7}{16} > 0$, hence, the decryption of 4-input OR gate upon $m_1 = m_2 = m_3 = m_4 = 1$ is 1, which is a correct decryption. Hence, following the above analysis, we observe that our proposed 4-input OR gate gives the correct output for all possible input combinations, proving this Boolean gate's correctness. □

Proof of Theorem 6.

Theorem 6: In TFHE, consecutive 3 binary XOR operations can not be integrated into one 4-input Boolean XOR operation, and therefore no such structure can be found for a 4-input XOR gate.

Proof. Consider a general representation of a 4-input XOR gate as follows,

$$\mathsf{XOR}_4(\mathbf{ct}_1, \mathbf{ct}_2, \mathbf{ct}_3, \mathbf{ct}_4) = \left(\frac{1}{c}, \mathbf{0}\right) + z_1 \cdot (\mathbf{ct}_1 + \mathbf{ct}_2) + z_2 \cdot \mathbf{ct}_3 + z_3 \cdot \mathbf{ct}_4.$$

Note that, the first two inputs should have the same scaling factor because the first two inputs correspond to the same (first) XOR gate. The remaining two are the alternate inputs of the second and third XOR gates in a consecutive structure.

From the definition of the TFHE decryption function, the phase computed during the decryption of a 4-input XOR can be written as,

$$\begin{aligned}\Phi = \frac{1}{c} + z_1 \cdot (\Delta(m_1) + \Delta(m_2)) + z_2 \cdot \Delta(m_3) + z_3 \cdot \Delta(m_4) +\\ z_1 \cdot (e_1 + e_2) + z_2 \cdot e_3 + z_3 \cdot e_4,\end{aligned}$$

or,

$$\Phi = \frac{1}{c} + \frac{1}{p_1} + \frac{1}{p_2} + \frac{1}{p_3} + \frac{1}{p_4} + \delta$$

.

Each $\frac{1}{p_i}$ denotes $z_i \cdot \Delta(m_{i+1})$ for $i > 1$ else $\frac{1}{p_i}$ denotes $z_i \cdot \Delta(m_i)$ and $\delta = z_1 \cdot (e_1 + e_2) + z_2 \cdot e_3 + z_3 \cdot e_4$. Here, we consider c, p_i $\forall i$ is > 0 and we alternate the sign according to the input. For example, if the first input if 0, we use $(-\frac{1}{p_1})$, otherwise use $(\frac{1}{p_1})$.

We know that the XOR operation returns true (or, 1) if the inputs have an odd number of 1's. Now, consider the following two cases–

Case 1. When three inputs are 1. Consider, the first input is 0 and rest three are 1, i.e.,

$$\Phi = \frac{1}{c} - \frac{1}{p_1} + \frac{1}{p_2} + \frac{1}{p_3} + \frac{1}{p_4} + \delta$$

To provide the correct output $\Phi > 0$ in this case, therefore, $\frac{1}{c} - \frac{1}{p_1} + \frac{1}{p_2} + \frac{1}{p_3} + \frac{1}{p_4} > 0$ as δ being small, or we can also write $\frac{1}{c} + \frac{1}{p_3} + \frac{1}{p_4} > 0$ as the first two inputs have same scaling factor thus canceling each other.

Case 2. When one inputs are 1. Consider, the first input is 1 and rest three are 0, i.e.,

$$\Phi = \frac{1}{c} + \frac{1}{p_1} - \frac{1}{p_2} - \frac{1}{p_3} - \frac{1}{p_4} + \delta$$

To provide the correct output $\Phi > 0$ in this case, therefore, $\frac{1}{c} + \frac{1}{p_1} - \frac{1}{p_2} - \frac{1}{p_3} - \frac{1}{p_4} > 0$ as δ being small, or we can also write $\frac{1}{c} - \frac{1}{p_3} - \frac{1}{p_4} > 0$. Now, this can not be possible because, $\frac{1}{c} < (\frac{1}{p_3} + \frac{1}{p_4})$. This can be proved with another input combination. Thus with contradiction, we prove that a 4-input XOR gate can not be constructed in the TFHE scheme.

Proof of $\frac{1}{c} < (\frac{1}{p_3} + \frac{1}{p_4})$. This we also prove with contradiction. Consider the case when the first two inputs are 1, and others are 0, i.e.,

$$\Phi = \frac{1}{c} + \frac{1}{p_1} + \frac{1}{p_2} - \frac{1}{p_3} - \frac{1}{p_4} + \delta$$

In this case, for correctness Φ should be less that 0, i.e., $\Phi < 0$. If, $\frac{1}{c} > (\frac{1}{p_3} + \frac{1}{p_4})$, the $\Phi > 0$, which will give a wrong decryption. Hence, by contradiction we proved $\frac{1}{c} < (\frac{1}{p_3} + \frac{1}{p_4})$.

The above case analysis finally proved our Theorem 7, i.e., in TFHE, consecutive 3 binary XOR operations can not be integrated into one 4-input Boolean XOR operation. □

B Description of Bristol Circuits

The Bristol format circuits are specifically designed for applications in MPC and FHE. The structure of a Bristol-represented circuit is shown in List 1. A Bristol Format file starts with a line that defines the number of Boolean gates and the number of wires in the Boolean circuit. The second line symbolizes the first two numbers as the number of input wires to the function with its corresponding number of output wires. Each line represents a binary gate and its input/output specifications from the third line onwards. The first two values represent the number of input and output respectively. The next two values are input wire numbers, which are fed as the inputs of the binary gate; after evaluation, the result is written in the next wire number, which is the output wire.

For example, in the 1^{st} line, *N_in_wire*, and *N_out_wire* denote the number of gates and the wires respectively. In 2^{nd} line, the first two values i.e., *in_size* refer to the size of the two inputs, and the third value *out_size* is the size of the output. Now, in 3^{rd} line the last

entry represents XOR operation taking the third and the fourth values i.e., in_wire_{11} and in_wire_{12} as input wires, and the result after homomorphic XOR operation is written in the output wire number out_wire_1. The first two values '2' and '1' denote the number of inputs and the number of outputs. The Bristol Formats are known to be TFHE-friendly as it is represented using only binary gates.

Listing 1: Bristol Format of a circuit

$N_in_wire \;\; N_out_wire$
$in_size \;\; in_size \;\; out_size$

$2 \;\; 1 \;\; in_wire_{11} \;\; in_wire_{12} \;\; out_wire_1 \;\; \mathsf{XOR}$
$2 \;\; 1 \;\; in_wire_{21} \;\; in_wire_{22} \;\; out_wire_2 \;\; \mathsf{XOR}$
$\vdots$
$2 \;\; 1 \;\; in_wire_{i1} \;\; in_wire_{i2} \;\; out_wire_i \;\; \mathsf{AND}$
$1 \;\; 1 \;\; in_wire_{j1} \;\; out_wire_j \;\; \mathsf{INV}$
$\vdots$

Algorithm 2 Our circuit to DAG conversion algorithm

Require: Boolean circuit Ckt consisting only binary gates.
Ensure: An adjacency matrix Adj representing an equivalent DAG for the input circuit Ckt.

Assume the total number of gates in the circuit is Num
$\mathsf{Adj}[Num+1][Num+1] \leftarrow 0$
$Num \leftarrow len(\mathsf{Ckt})$
for $i = 1; i \leq Num; i++$ **do**
    ▷ *Fill the first row and first column with the node numbers along with the gates they represent* ◁
    $\mathsf{Adj}[0][i] \leftarrow n_i^{G_i}$
    $\mathsf{Adj}[i][0] \leftarrow n_i^{G_i}$
for $i = 1; i \leq Num; i++$ **do**
    if $\mathsf{Ckt}[i][5] = XOR$ **then**
        $\mathsf{Adj}[\mathsf{Ckt}[i][2]][\mathsf{Ckt}[i][4]] \leftarrow 1$
        $\mathsf{Adj}[\mathsf{Ckt}[i][3]][\mathsf{Ckt}[i][4]] \leftarrow 1$

C Limitation of Hand-tuned Optimizations in TFHE for General Benchmarks

In this section, we presented specific examples of hand-tuned optimizations proposed in the literature highlighting their ability to scale for general benchmarks. Stracovsky et al. [Stracovsky et al.(2022)] presented methods to compute the following operations for AES efficiently: XOR, S-box, and multiplications on $\mathsf{GF}(2^8)$ elements[8] in the encrypted domain

[8] $\mathsf{GF}(2^8)$ represents Galois field operations on bytes.

Algorithm 3 Our circuit optimization algorithm

Require: An adjacency matrix Adj representation of the input circuit Ckt.
Ensure: An adjacency matrix Adj_{opt} representing a DAG for the optimized circuit Ckt_{opt}.

$Num \leftarrow len(\mathsf{Adj}[0]) - 1$
$tmp_set \leftarrow \{\}$
$\mathsf{Adj}_{opt} \leftarrow \mathsf{Adj}$
for $i = 1; i \leq Num; i++$ **do**
    $tmp \leftarrow \mathsf{Adj}_{opt}[i]$ ▷ *Extract the binary string for each row*
    if $hamming_weight(tmp) = 1$ **then**
        Extract the index j for which $\mathsf{Adj}_{opt}[i][j] = 1$
        if $\mathsf{Adj}_{opt}[j][0] = n_j^G$ && $G \neq X$ **then**
            if $\mathsf{Adj}_{opt}[i][0] = n_i^G$ **then**
                $tmp_set \leftarrow tmp_set \cup j$
                for $k = 1; k \leq Num; k++$ **do**
                    if $\mathsf{Adj}_{opt}[k][j] = 1$ && $k \neq i$ **then**
                        $\mathsf{Adj}_{opt}[k][i] \leftarrow 1$
                        $\mathsf{Adj}_{opt}[i][0] \leftarrow n_i^{GG}$
                        $\mathsf{Adj}_{opt}[0][i] \leftarrow n_i^{GG}$
                        for $l = 1; l \leq Num; l++$ **do**
                            if $\mathsf{Adj}_{opt}[j][l] \neq 0$ **then**
                                $\mathsf{Adj}_{opt}[i][l] \leftarrow \mathsf{Adj}_{opt}[j][l]$
        if $\mathsf{Adj}_{opt}[i][0] = n_i^G$ && $\mathsf{Adj}_{opt}[j][0] = n_j^X$ **then**
            $tmp_set \leftarrow tmp_set \cup j$
            for $k = 1; k \leq Num; k++$ **do**
                if $\mathsf{Adj}_{opt}[k][j] = 1$ && $k \neq i$ **then**
                    $\mathsf{Adj}_{opt}[k][i] \leftarrow 1$
                    $\mathsf{Adj}_{opt}[i][0] \leftarrow n_i^{GX}$
                    $\mathsf{Adj}_{opt}[0][i] \leftarrow n_i^{GX}$
                    for $l = 1; l \leq Num; l++$ **do**
                        if $\mathsf{Adj}_{opt}[j][l] \neq 0$ **then**
                            $\mathsf{Adj}_{opt}[i][l] \leftarrow \mathsf{Adj}_{opt}[j][l]$
Drop the f^{th} column and row from Adj_{opt} for every f in tmp_set and output the final optimized adjacency matrix Adj_{opt}.

using 8-bit to 8-bit Look-up-table (LUT) followed by FBS. They also utilize 16 threads to process 16 byte-elements in the state matrix in parallel and achieve 252 secs for homomorphic evaluation of AES-128. Trama et al. [Trama et al.(2023)] optimized AES evaluation using multi-value FBS with a specific choice of decomposition basis of ciphertext, that split a byte into two 4-bit messages, enabling smaller parameters. This technique achieves around 270 secs on single-thread and 28 secs on 16-threaded parallelism. Currently, the state-of-the-art implementation of AES is provided by Fregata [Wei et al.(2023)], which efficiently computes modular multiplication (required in MixColumns operation in AES) using simple XOR and shift operations. Fregata also used a mixed packing technique from [Chillotti et al.(2017)] for efficient S-box evaluation. Fregata can run AES-128 in 86 secs on a single core and in 9 secs with a 16-threaded parallel implementation. The mixed packing technique combines horizontal and vertical packing [Chillotti et al.(2017)] to pack all the outputs of an LUT into two ciphertexts, resulting in a faster LUT evaluation followed by an FBS. Therefore, mixed packing techniques are mostly useful for circuits having multi-input multi-output (preferably non-linear) substructures, which makes LUT evaluation and FBS efficient. Wang et al. [Wang et al.(2024)] further proposed a more compact and faster FBS, with efficient choices of parameters and provide results on AES-128, runs in 26 secs on single thread.

However, FBS is mostly beneficial for non-linear operations (such as S-box, ReLU, etc) or operations that have multi-input and multi-output structures. This is because FBS cost grows super-linearly with the fan-in size but is almost independent of the fan-out size of the operation [Carpov(2024)]. During AES evaluation, the intermediate operations (AddRoundKey, SubBytes, MixColumns) take a byte as input and output a byte. Hence, 8-bit to 8-bit LUTs followed by FBS are beneficial for such cases. But for circuits having mostly linear and sequential components with less fan-in and fan-out sizes are inefficient while using LUT-based implementation. This leads to a reduction in the overall efficiency of such circuits. In Section 4 we presented a detailed discussion on the performance of LUT-based designs and FBS for different structures of circuits.

On the other hand, these techniques also use 16 threads to parallelize the operations on the state matrix of AES. Such parallel computation can be scaled to other SIMD (Single Instruction Multiple Data) operations like Convolutions in neural networks but is not very useful for circuits having more sequential components such as Min/Max finding.

Therefore, circuits having predominantly low fan-in sequential components are expected to achieve better optimization using gate-based techniques, whereas circuits with high fan-in parallel components are more suitable for LUT-based optimization. Therefore, exploring a hybrid synthesis approach that combines both techniques could be an interesting direction. This would allow for leveraging the advantages of both methods, leading to a more efficient synthesis for circuits that contain a mix of (i) low fan-in sequential structures and (ii) high fan-in parallel structures.

D Our Circuit Synthesis on Bristol Circuits

In this section, we provide a detailed implementation of our circuit synthesis framework using a Bristol format circuit. Since any circuit can be converted into an equivalent Bristol representation, and because Bristol format circuits are convenient to process for FHE applications, our Algorithm 3 operates on Boolean circuits (represented in Bristol format) that

consist exclusively of binary gates and convert it into an equivalent DAG representation. Assume this input circuit has different binary gates such as AND, XOR, OR etc. Now, every binary gate is then labeled with a name corresponding to its operation, for example, the AND gates have the label "A", the XOR gates have the label "X", the OR gates have the label "O" etc. We also denote a binary gate with a node in a DAG, e.g., n_i^A denotes an AND gate at the i^{th} iteration in the circuit, similarly for other gates as well. Now, an edge $e_{p,q}$ is inserted in the graph directed from node n_p^G to $n_q^{G'}$ if and only if the output of a binary gate G at the p^{th} iteration is used as input to another binary gate G' at the q^{th} iteration. In more detail, line 2 in Algorithm 2 initializes an adjacency matrix Adj of dimension $(Num+1)\times(Num+1)$ with all zeros, where Num be the total number of binary gates present in the input circuit Ckt. Lines 4-7 initialize the placeholders of every node with the binary gate it represents, e.g., if $\mathsf{Adj}[i][0]$ and $\mathsf{Adj}[0][i]$ is $n_i^{G_i}$ then this node represent a binary G_i gate at the i^{th} iteration of the Bristol represented circuit Ckt. Finally, lines 8-13 put 1 in the cell $\mathsf{Adj}[i][j]$ if the output of a binary gate at the i^{th} iteration used as input to the binary gate at j^{th} iteration. This adjacency matrix Adj is then used as input to our circuit optimization algorithm mentioned in Algorithm 3.

Briefly, our circuit optimization algorithm (presented in Algorithm 3) is to find such a potential convex sub-graph and replace it with a more compact representation by combining two consecutive binary gates into a 3-input Boolean gate. We will now explain our circuit optimization algorithm in greater detail by walking through Algorithm 3 step by step. This algorithm takes an adjacency matrix Adj (output from Algorithm 2) as input and returns an optimized DAG represented circuit Adj_{opt}. Line 1 initializes Num to be the total number of binary operations in the circuit Ckt and line 2 initializes an empty set *tmp_set* which will be used to store the indexes of the second node in a convex sub-graph which can be reduced to a multi-input Boolean gate. This set will be finally used to remove those nodes from the optimized circuit after replacing all the potential multi-input gates. Line 3 first copies the entire matrix Adj into Adj_{opt}. Now, from line 4, we start iterating all the nodes in the new matrix Adj_{opt}, and line 5 stores each row as a binary string into a temporary variable *tmp*. Then in line 6, we check the Hamming weight (number of 1's) in the binary string *tmp*. Suppose, the Hamming weight of *tmp* is 1 for $tmp \leftarrow \mathsf{Adj}_{opt}[i]$, it then refers that the i^{th} node forms a convex sub-graph with a j^{th} node for which $\mathsf{Adj}_{opt}[i][j] = 1$. Now, lines 8-18 consider each convex sub-graph and check whether the second node is an XOR gate. If the second node represents a binary operation (say G) other than the XOR operation, then it checks if the first node also represents the binary operation G. If so, then it combines these two nodes and replaces them with a 3-input Boolean version of G. For that, the algorithm adjusts the adjacency matrix Adj_{opt} accordingly as presented by lines 11-18. Similarly, lines 19-28 deal with the convex sub-graphs where the second node represents an XOR operation. This case also encompasses the scenario of composite convex sub-graphs that is when the first node in the sub-graph represents a binary operation other than XOR; these convex sub-graphs are then replaced with an equivalent composite 3-input Boolean operation. Finally, the nodes present in the set *tmp_set* are removed from Adj_{opt} and the optimized Adj_{opt} matrix is converted into its equivalent circuit representation Ckt_{opt}. We have omitted the conversion algorithm from Adj_{opt} to Ckt_{opt}, as it is quite straightforward and similar to Algorithm 2 but in the opposite direction. The final optimized circuit Ckt_{opt} is homomorphically evaluated on encrypted inputs using the TFHE scheme and the evaluation time is presented in Table 3,4, and 5, where we demonstrate the efficiency of our proposed circuit synthesis algorithm by applying it across multiple benchmark circuits composed of

arithmetic, floating-point, and cryptographic functions.

E Super-linear Growth in FBS Cost

Bourse et al. [Bourse et al.(2020)] proves the following lemma,

Lemma 5. *The cost of functional bootstrapping (FBS) grows super-linearly with the input plaintext space.*

Bourse et al. [Bourse et al.(2020)] shows a probability of correctness after a bootstrapping as

$$erf\left(\left(\frac{1}{4B}-\frac{n+1}{4N}\right)\frac{1}{\sigma_{bs}\sqrt{2log_2(B)}}\right),$$

where, erf is the Gauss error function defined as $erf(z)=\frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$. This probability shows that increasing B requires to increase N, and the complexity of the bootstrapping is super-linear in N. Here, B is the plaintext space, N is the dimension of the Polynomial required in the blind rotation [Chillotti et al.(2020a)] phase during bootstrapping and σ_{bs} is the variance of the noise after a bootstrapping. For a more detailed discussion of this we refer to Section 2.5 in [Bourse et al.(2020)].