Towards Better Standard Cell Library: Optimizing Compound Logic Gates for TFHE

Kotaro Matsuoka
kmatsuoka@easter.kuee.kyoto-u.ac.jp
Kyoto University, Kyoto, Japan

Takashi Sato
takashi@i.kyoto-u.ac.jp
Kyoto University, Kyoto, Japan

Yusuke Hoshizuki
hoshizuki@axell.co.jp
AXELL CORPORATION, Tokyo, Japan

Song Bian
sbian@easter.kuee.kyoto-u.ac.jp
Kyoto University, Kyoto, Japan

ABSTRACT

In this paper, we cryptographically optimize the constructions of 3-input gates or multi-output gates using Fully Homomorphic Encryption (FHE) including Half Adder, Full Adder and AOI21. We implemented the gates over Torus Fully Homomorphic Encryption (TFHE). These gates achieve their respective functionalities with less computational cost than conventional constructions. By using the proposed gates, 16-bit adders and 16-bit multipliers can be evaluated about $5\times$ and $2.5\times$ faster, respectively, compared to the conventional constructions.

CCS CONCEPTS

• Security and privacy → Cryptography.

KEYWORDS

TFHE; Logic circuits; Full Adder

ACM Reference Format:

Kotaro Matsuoka, Yusuke Hoshizuki, Takashi Sato, and Song Bian. 2021. Towards Better Standard Cell Library: Optimizing Compound Logic Gates for TFHE. In *Proceedings of the 9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC '21)*, November 15, 2021, Virtual Event, Republic of Korea. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3474366.3486927>

1 INTRODUCTION

Fully Homomorphic Encryption (FHE) is a form of encryption that permits encrypted data to be evaluated by arbitrary functions without decryption [15]. In most FHE settings, ciphertexts contain noises because the encryption schemes rely on lattice problems like learning with errors (LWE) [14]. Meanwhile, homomorphic operations over such ciphertext increase the amount of noise. A greater level of noise in a ciphertext means a greater possibility of getting the wrong plaintext after the decryption. Therefore, the noise in the ciphertext needs to be reduced to permit further homomorphic evaluation. Bootstrapping [10] is a well-known technique to solve this problem by evaluating the decryption function over

the ciphertext, obtaining a fresh ciphertext with a noise level that is independent of previous homomorphic evaluations. The bootstrapping procedure is generally the most time-consuming part of FHE. By carefully setting up the parameters and FHE functionality, Torus Fully Homomorphic Encryption (TFHE) [6] achieves the state-of-the-art performance in evaluating the bootstrapping procedure, at a speed of around 10 ms. TFHE natively supports homomorphic logic operations, so the scheme is suitable for evaluating homomorphic functions formulated as logic circuits.

Logic circuits have been studied intensively in the semiconductor industry. Therefore, using the mature know-how and tools of the semiconductor industry is an obvious way to create faster and more optimized logic circuit representations of functions over TFHE. For example, Yosys [16], one of the open-source logic synthesis suites, is used in Cingulata [4] and VSP [13] to construct logic circuits over TFHE. The standard cell methodology is one kind of semiconductor design technique that represents the device by the standard cells instead of directly placing transistors. The standard cell library is a set of small circuits which represent the frequently used Boolean functions like NAND, MUX, Full Adder, etc. In analogy to semiconductor design, the basic operation of FHE can be viewed as an elementary circuit composed of certain types of transistors. This is because that in semiconductor design, complex logic circuits are expressed as a combination of such elementary circuits, while in FHE, complex operations are expressed as a combination of the basic operations. Therefore, designing cryptographically optimized representations of small Boolean functions as a standard library for FHE enables the application of further circuit optimization by existing logic synthesis tools.

To build a better standard cell library for FHE, we propose some 3-input gates and multi-output gates. Our proposed gates are cryptographically optimized so that the same logic function can be evaluated with less computational effort than the conventional gates [6]. Our proposed gates can be divided into two categories by the encoded message spaces they use: conventional and extended (denoted as $\mathbb{M}_8$ and $\mathbb{M}_{12}$, respectively). The conventional encoded message space is more suitable for integrating with the conventional gates [6] while the extended one gives more performance benefits. Our evaluations show that our proposed gates result in better gate-level performance, and that logic synthesis tools, e.g., Yosys [16], can successfully exploit the performance benefits provided by our proposed gates in real circuits.

This work is licensed under a Creative Commons Attribution International 4.0 License.

WAHC '21, November 15, 2021, Virtual Event, Republic of Korea.

© 2021 Copyright is held by the owner/author(s).

ACM ISBN 978-1-4503-8656-2/21/11.

<https://doi.org/10.1145/3474366.3486927>

2 TFHE

In this section, we explain basic concepts of TFHE [6] and the optimizations for TFHE we used to construct our proposed gates.

2.1 Notations

$\mathbb{B}$: The set $\{1, 0\} \subset \mathbb{Z}$.

$\mathbb{T}$: The real Torus $\mathbb{R}/\mathbb{Z}$, i.e., the set of real number modulo 1. In this work, we define the interval of $\mathbb{T}$ to be $[-0.5, 0.5)$.

n, N : n is the dimension of the TLWElv0 ciphertext. N is the dimension of TLWElv1 and the degree of polynomials of TRLWE. Here, we take N to be a power of two.

$\mathbb{T}_N[X], \mathbb{Z}_N[X], \mathbb{B}_N[X]$: The rings of polynomials $\mathbb{R}[X]/(X^N + 1) \bmod 1$, $\mathbb{Z}[X]/(X^N + 1)$ and $\mathbb{B}[X]/(X^N + 1)$.

$1[X]$: The polynomial in $\mathbb{Z}_N[X]$ whose coefficients are all 1.

$m, m[X]$: The message. $m \in \mathbb{B}, m[X] \in \mathbb{B}_N[X]$.

$\mathbb{M}_t$: The encoded message space. $\mathbb{M}_t = \{-\frac{1}{t}, \frac{1}{t}\} \subset \mathbb{T}$. In this work, we use $\mathbb{M}_8$ or $\mathbb{M}_{12}$.

$\mathbb{M}_t^-, \mathbb{M}_t^+$: $\mathbb{M}_t^-$ is the negative element of $\mathbb{M}_t$, corresponding to message 0. For example, $\mathbb{M}_8^-$ is $-\frac{1}{8}$. $\mathbb{M}_t^+$ is the positive element corresponding to message 1.

$\text{sgn}(\cdot)$: $\text{sgn}(x) = 1$ if $x > 0$, and equals to -1 otherwise.

E^p : The set of vectors of dimension p with entries in E .

$U(E)$: The uniform distribution over E .

$\mathcal{D}_{\mathbb{T}, \sigma}$: Modular Gaussian distribution [6] over Torus with the variance $\sigma \in \mathbb{R}^+$.

$\leftarrow$: $x \leftarrow D$ means x itself or its entries or coefficients are drawn from the distribution D .

$a[X], b[X]$: $a[X], b[X] \in \mathbb{T}_N[X]$.

l, Bg : $l, Bg \in \mathbb{N}$ are parameters for TRGSW.

s, S : The secret keys for TLWElv0 and TLWElv1 ciphertexts respectively. They are drawn from $U(\mathbb{B})$.

$S[X]$: The secret key for TRLWE and TRGSW ciphertexts. It is a polynomial in $\mathbb{B}_N[X]$ and its i th coefficient is i th element of S .

BK: Bootstrapping Key; i.e., a list of TRGSW ciphertexts, each one encrypting a bit of the secret key s .

IKSK: Identity Key Switching Key; i.e., a list of TLWElv0 ciphertexts, each one encrypting a bit of the secret key S .

2.2 Forms of Ciphertexts

In this subsection, we explain the three forms of ciphertexts used in TFHE.

2.2.1 TLWE. TLWE is the Torus version of LWE [14]. TLWE ciphertext can be represented as (a, b) , an $n + 1$ (TLWElv0) or $N + 1$ (TLWElv1) dimensional Torus vector. In this paper, we focus on Boolean functions, so the message m is in $\mathbb{B}$. However, because a TLWE ciphertext is a Torus vector, the message must be represented by an element of $\mathbb{T}$, not $\mathbb{B}$. Hence, we use $\mathbb{M}_t$ to encode the message in $\mathbb{T}$. In the following, we describe the encryption and decryption equations of TLWElv0. Replace n, s by N, S for TLWElv1.

Encryption: Let $e \leftarrow \mathcal{D}_{\mathbb{T}, \sigma}(x)$ and $\mathbb{T}^n \ni a \leftarrow U(\mathbb{T})$. Then, $b = a \cdot s + \mathbb{M}_t^+ \cdot (2m - 1) + e$.

Decryption: Return $(1 + \text{sgn}(b - a \cdot s))/2$.

2.2.2 TRLWE. TRLWE is the Torus version of ring-LWE. TRLWE can be represented as $(a[X], b[X])$, a 2 dimensional Torus polynomial vector.

Encryption: Let $e[X] \in \mathbb{T}_N[X] \leftarrow \mathcal{D}_{\mathbb{T}, \sigma}(x)$ and $a[X] \leftarrow U(\mathbb{T})$. Then, $b[X] = a[X] \cdot S[X] + \mathbb{M}_t^+ \cdot (2 \cdot m[X] - 1[X]) + e[X]$.

Decryption: Return $(1[X] + \text{sgn}(b[X] - a[X] \cdot s[X]))/2$.

2.2.3 TRGSW. This is a Torus and ring version of GSW [11]. We omit the details about TRGSW because this form is only used for representing **BK**.

2.3 TFHE Primitives

In this subsection, we explain the components needed for the construction of gates proposed in the original TFHE paper [6]. These are also used in our gates.

2.3.1 Blind Rotate. In this paper, Blind Rotate (BR) refers the operation that allows for the homomorphic evaluation of a negacyclic rotation over the coefficients of the input plaintext polynomial $TV[X]$. BR takes a TLWElv0 ciphertext $cin = (a, b)$, the Bootstrapping Key **BK** and a test vector (the input plaintext polynomial) $TV[X]$ as inputs. Let $\rho = \lfloor 2N \cdot b \rfloor - \sum_{i=0}^{n-1} \lceil 2N \cdot a_i \rceil \cdot s_i \bmod 2N$. BR homomorphically evaluates the following:

$$\begin{aligned} &\text{BlindRotate}(cin, \mathbf{BK}, TV[X]) \\ &= (\text{TRLWE ciphertext encrypting } X^{-\rho} \cdot TV[X]). \end{aligned} \quad (1)$$

The graphical image of the correspondence between the plaintext of the input TLWElv1 ciphertext and ρ is given in Figure 1.

2.3.2 Sample Extract Index. This operation extracts one coefficient of the input TRLWE ciphertext $trlwe$ specified by the given index i and returns the extracted coefficient as a TLWElv1 ciphertext. This operation does not increase the noise.

$$\begin{aligned} &\text{SampleExtractIndex}(trlwe, i) \\ &= (\text{TLWElv1 encrypting the } i\text{th coef. of the message}). \end{aligned} \quad (2)$$

2.3.3 Identity Key Switching. Identity Key Switching (IKS) denotes the special case of Public Key Switching [6] where the public function is the identity function. This operation takes a TLWElv1 ciphertext $clvl1$ and Identity Key Switching Key **IKSK**, and returns a TLWElv0 ciphertext encrypting the plaintext of the input TLWElv1 ciphertext.

$$\begin{aligned} &\text{IdentityKeySwitching}(clvl1, \mathbf{IKSK}) \\ &= (\text{TLWElv0 ciphertext encrypting the plaintext of } clvl1). \end{aligned} \quad (3)$$

2.3.4 Programmable Bootstrapping. Programmable Bootstrapping refers to evaluating Look Up Table (LUT) over TFHE by BR. The constant term of $X^{-\rho} \cdot TV[X]$, $(X^{-\rho} \cdot TV[X])_0$, is determined as the following:

$$(X^{-\rho} \cdot TV[X])_0 = \begin{cases} (TV[X])_{\rho} & \text{if } \rho \in [0, N) \\ -(TV[X])_{\rho-N} & \text{if } \rho \in [N, 2N). \end{cases} \quad (4)$$

For example, in the $N = 16$ case, $(TV[X])_1 = -(TV[X])_{17}$ and $\rho = 1, 17$ are in the opposite position in Figure 1. This equation means that the combination of BR and Sample Extract Index of the constant term can be seen as the evaluation of LUT whose index is ρ and which returns $(X^{-\rho} \cdot TV[X])_0$.

Remark 1. Though ρ takes $2N$ values, $(X^{-\rho} \cdot TV[X])_0$ takes N coefficients of $TV[X]$ and its negative sign ones. This constraint

Figure 1: Correspondance between the plaintext and ρ
 $(\mathbb{M}_8 = \{-\frac{1}{8}, \frac{1}{8}\}, N=16)$

Algorithm 1 HomGate

Require: ca, cb : TLWElv0, IKSK, BK

Require: $scale, offset$: Parameters to construct a specific gate.

$cadd \leftarrow scale \cdot (ca + cb) + offset$

$crot \leftarrow \text{BlindRotate}(cadd, \text{BK}, \sum_{i=0}^{N-1} \frac{1}{8} X^i)$

$cylvl1 \leftarrow \text{SampleExtractIndex}(crot, 0)$

$cy \leftarrow \text{IdentityKeySwitching}(cylvl1, \text{IKSK})$

return cy : TLWElv0

means that we have only N degrees of freedom to construct LUT which have $2N$ possible outputs.

Remark 2. In this paper, we do not use all N degrees of freedom in the coefficients selection of $TV[X]$ because we only treats two cases of encoded message spaces, namely $\mathbb{M}_8$ and $\mathbb{M}_{12}$. This constraint implies that possible values of ρ is limited because the number of possible plaintexts of the input to BR is limited to $t/2$ and ρ is $2N$ times the plaintext values plus noise and rounding error. In addition to that, we want to make sure that the noise and rounding errors do not affect the evaluation results of the LUT. Therefore, we divide the N degrees of freedom into $t/4$ continuous segments and keep all LUT values placed in the same segment the same. In other words, we only use $t/4$ degrees of freedom.

In Figure 1, we take the $\mathbb{M}_8$ case as an example to better explain the above two remarks. In the $\mathbb{M}_8$ case, the $2N$ possible ρ values are divided into $t/2 = 4$ segmented pies. The values at the center of each segmented pie show corresponding plaintexts of input ciphertexts. As noted in Remark 2, while we have $2N$ ρ values, we only have $t/4 = 2$ degrees of freedom here, namely the two segments: $[0, N/2)$ and $[N/2, N)$. The LUT values of the other two segments are negation of these two segments as noted in Remark 1.

2.4 Constructions of Conventional Gates

In this paper, we use $\mathbb{M}_8$ as the message space for conventional gates because it is used in the original implementation [7]. The 2-input-1-output gates proposed in the original TFHE paper [6] share the same algorithm as Algorithm 1. The only differences are the parameter inputs, $scale$ and $offset$. The parameter inputs for each gate are given in Table 1. The test vector used in Algorithm 1

Table 1: Parameters for 2-input gates

Parameter	AND	OR	XOR	NAND	NOR	XNOR
$scale$	1	1	2	-1	-1	-2
$offset$	$-\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$	$-\frac{1}{8}$	$-\frac{1}{4}$

can be seen as the LUT to evaluate the $sgn(\cdot)$ function, which is used in the decryption procedure.

Remark 3. NOT can be constructed by just negating all coefficients of the input ciphertext.

2.5 Primitives used in Multi-output Gates

Here, we describe the building blocks used in the constructions of our multi-output gates, namely, Gao's method [9] and Multi-value Bootstrapping [5].

2.5.1 Gao's method. In [9], Gao constructed Fully Homomorphic Encryption based on FHEW [8] which evaluates AND, OR, XOR by one bootstrapping procedure. Because TFHE is the successor of FHEW, we extend some ideas that Gao used to evaluate three gates at once (we call these ideas Gao's method) to TFHE in Section 3.2.1.

2.5.2 Multi-value Bootstrapping. Multi-value Bootstrapping [2, 5] is the way to evaluate multiple LUTs by one BR. There are some variations of Multi-value Bootstrapping [5] and we used the one described in [2]. The idea of Multi-value Bootstrapping we use in this paper is to extract the common factor $TV_0[X]$ of multiple test vectors and only apply BR to $TV_0[X]$. If we assume the test vector $TV[X]$ can be represented as $TV[X] = TV_s[X] \cdot TV_0[X]$, it holds that $X^{-\rho} \cdot TV[X] = TV_s[X] \cdot X^{-\rho} \cdot TV_0[X]$. Therefore, evaluating $X^{-\rho} \cdot TV_0[X]$ by BR and multiplying $TV_s[X]$ to it gives the TRLWE ciphertext where the plaintext is the same as $X^{-\rho} \cdot TV[X]$. Since $TV_0[X]$ is the common factor, the result of BR, $X^{-\rho} \cdot TV_0[X]$, can be reused. Using different $TV_s[X]$ vectors then corresponds the evaluations of different LUTs. For example, in the $\mathbb{M}_{12}$ case, we can control only $t/4 = 3$ continuous segments. This constraint means that, because each segment only takes values in $\mathbb{M}_{12}$, any test vector can be expressed as

$$\left(p_0 + p_1 X^{\lfloor \frac{N}{3} \rfloor} + p_2 X^{2\lfloor \frac{N}{3} \rfloor} \right) \sum_{i=0}^{\lfloor \frac{N}{3} \rfloor - 1} \frac{1}{12} \quad (5)$$

Where $p_i \in \{-1, 1\}$. Therefore, we can factor out $\sum_{i=0}^{\lfloor \frac{N}{3} \rfloor - 1} \frac{1}{12}$ as the common factor in this case. In [2], Biasse uses the above technique to evaluate Full Adder (FA) over FHEW, and we extended the idea to TFHE in Section 3.3.1.

3 PROPOSED GATES

In this section, we explain the constructions of our proposed gates. We propose multi-output or 3-input gates. The gates other than 2-input NAND in Table 4 are our proposed gates. All of our multi-output gates can be represented by multiple 1-output gates sharing the same input like Half Adder (HA) and FA. This section is consisted of three subsections. Section 3.1 explains why we change the order of BR and IKS. Sections 3.2 and 3.3 explain the constructions of our proposed gates using $\mathbb{M}_8$ and $\mathbb{M}_{12}$, respectively.

Algorithm 2 Half Adder

Require: ca, cb : TLWElv1, IKSK, BK
 $cadd \leftarrow \text{IdentityKeySwitching}(ca + cb, \text{IKSK})$
 $crot \leftarrow \text{BlindRotate}(cadd, \text{BK}, \sum_{i=0}^{N-1} \frac{1}{8} X^i)$
 $cand \leftarrow -\text{SampleExtractIndex}(crot, N - 2N \cdot \frac{1}{8})$
 $cor \leftarrow \text{SampleExtractIndex}(crot, 2N \cdot \frac{1}{8})$
 $cxor \leftarrow cor - cand - \frac{1}{8}$
return $cxor, cand$: TLWElv1

Table 2: Correspondence between input ciphertexts and an output ciphertext of OR, AND and XOR gates

(a) OR			(b) AND			(c) XOR		
$ca \backslash cb$	$-\frac{1}{8}$	$\frac{1}{8}$	$ca \backslash cb$	$-\frac{1}{8}$	$\frac{1}{8}$	$ca \backslash cb$	$-\frac{1}{8}$	$\frac{1}{8}$
$-\frac{1}{8}$	$-\frac{1}{8}$	$\frac{1}{8}$	$-\frac{1}{8}$	$-\frac{1}{8}$	$-\frac{1}{8}$	$-\frac{1}{8}$	$-\frac{1}{8}$	$\frac{1}{8}$
$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$-\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$-\frac{1}{8}$

3.1 Moving IKS Around

In our construction of the proposed gates, we used TLWElv1 as the input and output ciphertext formats of the gates. This modification allows us to exchange the order of BR and IKS. This technique is also used in [3]. Exchanging the order gives us two benefits. The first benefit is the reduction of the noise. The addition of TLWElv1 ciphertexts introduces less noise than TLWElv0 ones because TLWElv0 ciphertexts are produced by applying IKS to TLWElv1 ciphertexts, and IKS increases the noise. The second benefit is the reduction of the number of IKS in the gates. In a multi-output gate, if TLWElv0 is adopted as the output ciphertext format, it means that IKS must be done for as many outputs as there are. In contrast, if TLWElv1 is adopted, only one IKS is enough. Because of the larger ciphertext size compared to TLWElv0, using TLWElv1 as the input and output ciphertext format increases the memory consumption [3]. However, we believe the two benefits listed above can make up for the extra cost.

3.2 Gates Using $\mathbb{M}_8$

In this subsection, we explain the construction of our proposed gates which use $\mathbb{M}_8$ as the encoded message space. Since this space is also used in conventional gates, gates proposed here can be easily integrated into the circuits consisted of conventional gates.

3.2.1 Half Adder and 2-input Gates. HA is the gate constructed by XOR and AND gates sharing the same inputs. HA computes the addition of 2 bits and is one of the fundamental and frequently used gates in the construction of basic circuits. The pseudo-code of HA is given in Algorithm 2.

Two ideas are needed to understand the construction. (1): XOR can be represented by the subtraction of the output of AND and the constant $\frac{1}{8}$ from the output of OR. (2): The only difference between AND and OR is the amount of rotation in BR.

(1): Let ca and cb be the input ciphertexts, and let cor , $cand$ and $cxor$ be the output ciphertext of OR, AND and XOR gates, respectively. Table 2 show the correspondence between plaintexts

Algorithm 3 2BR Full Adder

Require: ca, cb, cc : TLWElv1, IKSK, BK
 $cadd \leftarrow \text{IdentityKeySwitching}(ca + cb + cc, \text{IKSK})$
 $cxorrot \leftarrow \text{BlindRotate}(2 \cdot cadd, \text{BK}, \sum_{i=0}^{N-1} \frac{1}{8} X^i)$
 $cxor \leftarrow \text{SampleExtractIndex}(cxorrot, 0)$
 $ccarryrot \leftarrow \text{BlindRotate}(cadd, \text{BK}, \sum_{i=0}^{N-1} \frac{1}{8} X^i)$
 $ccarry \leftarrow \text{SampleExtractIndex}(ccarryrot, 0)$
return $cxor, ccarry$: TLWElv1

of ca , cb and the plaintexts of output ciphertexts. The equation $cxor = cor - cand - \frac{1}{8}$ is easily verified by inserting the plaintext values of the output ciphertexts shown in Table 2.

(2): By comparing the corresponding columns of Table 1, we can see that the only difference between AND and OR is the *offset*. In the evaluation of 2-input gates, BR evaluates

$$X^{-(\lfloor 2N \cdot (\text{offset} + ca.b + cb.b) \rfloor - \sum_{i=0}^{N-1} \lfloor 2N \cdot (ca.a_i + cb.a_i) \rfloor \cdot s_i)} \cdot TV[X] \quad (6)$$

Because $2N \cdot \text{offset}$ is always an integer in realistic security parameters, we can factor out the $X^{-\text{offset}}$ term and obtain below:

$$X^{-2N \cdot \text{offset}} \cdot X^{-(\lfloor 2N \cdot (ca.b + cb.b) \rfloor - \sum_{i=0}^{N-1} \lfloor 2N \cdot (ca.a_i + cb.a_i) \rfloor \cdot s_i)} \cdot TV[X] \quad (7)$$

Equation (7) implies that we can share the result of BR for $ca + cb$, where the evaluation of AND and OR respectively corresponds to multiplying $X^{-2N \cdot \frac{1}{8}}$ and $X^{-2N \cdot \frac{1}{8}}$ to the result of BR. In the actual algorithm shown in Algorithm 2, we change the input i of Sample Extract Index instead of the multiplication of $X^{-\text{offset}}$ to avoid unnecessary operations.

Other 2-input Gates sharing the same inputs: The evaluation of XOR in HA uses the result of OR. Therefore, just returning it gives the construction of the gate which evaluates AND, OR, and XOR at once like the original Gao's method [9]. In addition to that, since NOT is just a negation of the elements of a ciphertext, NAND, NOR and XNOR also can be evaluated at once.

3.2.2 2BR Full Adder. FA is the gate constructed by a 3-input XOR gate and a majority (carry) gate sharing the same inputs. The pseudo-code is given in Algorithm 3. FA computes the addition of 3 bits, so this is also the fundamental and frequently used gate. If we construct FA using HAs, two HAs and one OR gate are needed. The evaluation of these gates needs three BRs and these gates must be evaluated in serial. Two Blind Rotate Full Adder achieves the functionality of an FA by evaluating a 3-input XOR gate and a majority gate in parallel. Let ca, cb and $ccarryin$ be input ciphertexts.

3-input XOR: The construction of 3-input XOR follows the construction of 2-input XOR [6]. The plaintexts of $2 \cdot (ca + cb + ccarryin)$ is $\frac{1}{4}$ if the plaintext of the odd number of the ciphertexts are $\frac{1}{8}$, and is $-\frac{1}{4}$ otherwise. Therefore, the evaluation of BR with $TV[X] = \sum_{i=0}^{N-1} \frac{1}{8} X^i$ corresponds to the evaluation of 3-input XOR.

Majority: The plaintext of $ca + cb + ccarryin$ is a positive number, $\frac{1}{8}$ or $\frac{3}{8}$, if the plaintexts of two or three input ciphertexts are $\frac{1}{8}$ and is a negative number, $-\frac{1}{8}$ or $-\frac{3}{8}$, otherwise. Therefore, the evaluation of BR with the same test vector corresponds to the evaluation of the majority gate.

Algorithm 4 1BR Full Adder

Require: $ca, cb, ccarryin$: TLWEvl1, IKSK, BK
 $cadd \leftarrow \text{IdentityKeySwitching}(ca + cb + ccarryin, iksk)$
 $TV_{xor}[X] \leftarrow -1 + X^{\lfloor \frac{N}{3} \rfloor} - X^{2 \cdot \lfloor \frac{N}{3} \rfloor}$
 $TV_{carry}[X] \leftarrow 1 + X^{\lfloor \frac{N}{3} \rfloor} + X^{2 \cdot \lfloor \frac{N}{3} \rfloor}$
 $TV_0[X] \leftarrow \sum_{i=0}^{\lfloor \frac{N}{3} \rfloor - 1} \frac{1}{12} X^i$
 $crot \leftarrow \text{BlindRotate}(cadd, \text{BK}, TV_0)$
 $cxor \leftarrow \text{SampleExtractIndex}(TV_{xor}[X] \cdot crot, 0)$
 $ccarryout \leftarrow \text{SampleExtractIndex}(TV_{carry}[X] \cdot crot, 0)$
return $cxor, ccarryout$: TLWEvl1

3.3 Gates Using $\mathbb{M}_{12}$

In this section, we explain the constructions of our proposed gates which use $\mathbb{M}_{12}$ as the encoded message space. Using this narrower encoded message space enables us to evaluate some 3-input gates in exchange for increased decryption error probability. Connecting the gates using $\mathbb{M}_{12}$ with ones using $\mathbb{M}_8$ incurs increased noise levels because the connection needs the multiplication of the input ciphertexts a constant of $\frac{3}{2}$ (or some other modification equivalent to this multiplication). Therefore, mixing these gates with conventional gates needs additional consideration about the noise.

3.3.1 1BR Full Adder and 3-input Gates. As the name suggests, 1BR Full Adder achieves the functionality of an FA by evaluating one BR while the 2BR Full Adder described in Section 3.2.2 needs two BRs. The pseudo-code is given in Algorithm 4. We use the sum of the three input TLWEvl1 ciphertexts as the input for BR in this construction. In this case, Gao's method [9] cannot construct a 3-input XOR because such gate cannot be represented by the linear combination of the results of AND and OR. Therefore, we use Multi-value Bootstrapping [2, 5] described in Section 2.5.2 to evaluate two LUTs for 3-input XOR and the majority gate by one BR. Figure 2 illustrates the evaluation of 3-input XOR in the $N = 16, \rho = 8$ case. The arrays of numbers with boxes represents coefficients of polynomials of $TV_0[X]$, $X^{-\rho} \cdot TV_0[X]$ and $TV_{xor}[X] \cdot X^{-\rho} \cdot TV_0[X]$. The number in the lower left corner with the gray box is the output, $(TV_{xor}[X] \cdot X^{-\rho} \cdot TV_0[X])_0$.

Other 3-input Gates sharing the same inputs: Like HA in the $\mathbb{M}_8$ case, 3-input AND and OR can be evaluated at once with FA because the difference between these two gates and the majority gate can be represented by the constant negacyclic rotation in BR evaluation. We also note that NOT of the results can also be evaluated at once.

3.3.2 AOI21 and OAI21. By modifying the construction of 1BR FA, we are also able to construct AOI21, one kind of 3-input gates. The pseudo-code is shown in Algorithm 5. AOI21 is a bit different from the other proposed gates because it is the gate represented by the chain of an AND and a NOR gate. Let $a, b, c \in \mathbb{B}$ are the inputs of AOI21. Then, AOI21 can be represented as $\neg((a \wedge b) \vee c)$. Table 3 shows the truth table of AOI21. There are two important ideas to construct AOI21. (1): c needs to be treated differently from a and b . (2): we use only five continuous segments of the test vector.

(1): Because we restrict the input of BR to a linear combination of ciphertexts with integer coefficients, c can be treated differently from a and b by tweaking the coefficients in the linear combination.

Algorithm 5 AOI21

Require: ca, cb, cc : TLWEvl1, IKSK, BK
 $cadd \leftarrow \text{IdentityKeySwitching}(ca + cb + 2 \cdot cc + \frac{1}{12}, \text{IKSK})$
 $crot \leftarrow \text{BlindRotate}(cadd, \text{BK}, \sum_{i=0}^{N-1} -\frac{1}{12} X^i)$
 $caoi \leftarrow \text{SampleExtractIndex}(crot, 0)$
return $caoi$: TLWEvl1

Table 3: Truth table for AOI21

$c \backslash ab$	00	01	11	10
0	1	1	0	1
1	0	0	0	0

Let ca, cb and cc be TLWEvl1 ciphertexts of the inputs a, b and c , respectively. Then, we use $ca + cb + 2 \cdot cc + \frac{1}{12}$ as the linear combination to achieve treating c in a different way. The constant $\frac{1}{12}$ is added to simplify the test vector.

(2): The plaintext of $ca + cb + 2 \cdot cc + \frac{1}{12}$ is in $\{-\frac{1}{4}, -\frac{1}{12}, \frac{1}{12}, \frac{1}{4}, \frac{5}{12}\}$. The colors of cells in Table 3 depict plaintext values of $ca + cb + 2 \cdot cc + \frac{1}{12}$, where the same color corresponds to the same plaintext value. For example, the plaintext value corresponds to $(a, b, c) = (0, 1, 0)$, $(1, 0, 0)$ is $-\frac{1}{12}$. This plaintext value is the negation of the plaintext that corresponds to $(1, 0, 1)$, $(0, 1, 1)$, and the output is also the negation of the one corresponding to $(0, 1, 0)$, $(1, 0, 0)$. The same relationship holds between $(0, 0, 0)$ and the pair of $(0, 1, 1)$ and $(1, 0, 1)$. This negation relationship implies that the constraint described in Remark 1 holds. Since there are only five colors in Table 3, using five continuous segments is enough to represent the LUT for AOI21. Three segments are directly controlled by the $t/4 = 3$ degree of freedom of the test vector and the other two segments are the negation of the three segments.

OAI21: Because OAI21, $\neg((a \vee b) \wedge c)$, is very similar to AOI21, we can construct it by multiplying $X^{\lfloor \frac{N}{3} \rfloor}$ to $crot$ in Algorithm 5.

4 EVALUATION

In this section, we give evaluations of our proposed gates.

4.1 Real Performance

In this subsection, we give the performance evaluation of our implementation¹ (based on TFHEpp [12]) of the proposed gates on one core of AMD Ryzen 3700X processor. We use parameters $n = 635, N = 1024, l = 3, Bg = 2^6$. This parameter is estimated to achieve 128-bit security by lwe-estimator [1]. Each gate is evaluated 100 times and the mean runtime is reported in Table 4. The values after $\pm$ represent the standard deviations. Our proposed gates other than 2BR Full Adder take slightly longer time to evaluate in comparison to 2-input NAND because the handling of 3-input or multi-output needs additional operations. The 2BR Full Adder takes roughly twice as long as the other gates because the gate needs the evaluation of two BRs. Because HA, FA and OAI21 need two, five and two BRs respectively when implemented by conventional gates, all of our proposed gates achieve their respective functionalities with less computational cost than the conventional constructions.

¹Our implementation will be available on <https://github.com/axell-corp/oveus-tfhe>

Figure 2: Multivalue Bootstrapping for 3-input XOR ($N = 16, \rho = 8$)

Table 4: Performance of the gates

Gate Type	2-input NAND	Half Adder	2BR Full Adder
time[ms]	11.66 ± 0.15	12.13 ± 0.11	23.7 ± 1.5
Gate Type	1BR Full Adder	AOI21	
time[ms]	11.96 ± 0.13	12.06 ± 0.33	

Table 5: Results of synthesis by Yosys

Circuit(16-bit inputs)	Adder	Multiplier
# of BRs (conventional)	83	1489
# of BRs ($\mathbb{M}_8$)	32	812
# of BRs ($\mathbb{M}_{12}$, w {AO,OA}I21)	16	641
# of BR ($\mathbb{M}_{12}$, w/o {AO,OA}I21)	16	601

4.2 Synthesis

In this subsection, we give the results of logic synthesis by Yosys [16] to illustrate how much our proposed gates are beneficial in the synthesized circuits. We show the number of BRs for a 16-bit adder and a 16-bit multiplier in Table 5, since the number of BRs is almost proportional to the total run-time for evaluating the circuit by a single thread of CPU. We note that the number of BRs alone does not completely determine the total run-time in a multi-core environment because some gates can be evaluated in parallel. In both circuits, the worst case is the case only using conventional gates and the best case is the case only using $\mathbb{M}_{12}$ case without AOI21 and OAI21. The reason why using AOI21 and OAI21 increase the number of BRs is that we synthesized this circuit by the default constraints of Yosys, so the synthesis procedure is not fully optimized for TFHE. In particular, the number of BRs are different between each gates but this constraint is not taken into consideration during the synthesis process.

4.3 Planned DEMO

During the demo, we plan to give a brief overview of the construction of our gates and implementations. After that, we will show the workflow to use our gates in circuit evaluation on real environments in cooperation with other open-source tools including Yosys.

5 CONCLUSION

Our cryptographically optimized 3-input gates and multi-output gates over TFHE offer a new way to evaluate compound logical operations with less computational cost compared to conventional constructions. We demonstrate that using such optimized gates,

the performance of basic circuit operations (e.g., Full Adder) can be greatly improved. In addition, the designed gates can be integrated into logic synthesis tools to fully exploit their benefits, and our gates can be seen as an effort to build better standard cell library for TFHE.

6 ACKNOWLEDGEMENT

This work was partially supported by JSPS KAKENHI Grant No. 20K21793 and JST CREST Grant No. JPMJCR19K5.

REFERENCES

- [1] 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. <https://doi.org/doi:10.1515/jmc-2015-0016> <https://bitbucket.org/malb/lwe-estimator/src/fb7deba/>.
- [2] Jean-François Biasse and Luis Ruiz. 2015. FHEW with Efficient Multibit Bootstrapping. In *Progress in Cryptology – LATINCRYPT 2015*, Kristin Lauter and Francisco Rodríguez-Henríquez (Eds.). Springer International Publishing, Cham, 119–135.
- [3] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *Advances in Cryptology – CRYPTO 2018 (Lecture Notes in Computer Science)*, Vol. 10993. Springer, 483–512. https://doi.org/10.1007/978-3-319-96878-0_17
- [4] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. 2015. Armadillo: a compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*. 13–19.
- [5] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In *Topics in Cryptology – CT-RSA 2019*, Mitsuru Matsui (Ed.). Springer International Publishing, Cham, 106–126.
- [6] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33, 1 (01 Jan 2020), 34–91. <https://doi.org/10.1007/s00145-019-09319-x>
- [7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. August 2016. TFHE: Fast Fully Homomorphic Encryption Library. (August 2016). <https://tfhe.github.io/tfhe/>.
- [8] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *Advances in Cryptology – EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 617–640.
- [9] Shuhong Gao. 2018. Efficient Fully Homomorphic Encryption Scheme. Cryptology ePrint Archive, Report 2018/637. (2018). <https://eprint.iacr.org/2018/637>.
- [10] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Ph.D. Dissertation. Stanford University. crypto.stanford.edu/craig.
- [11] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. Cryptology ePrint Archive, Report 2013/340. (2013). <https://eprint.iacr.org/2013/340>.
- [12] Kotaro Matsuoka. 2020. TFHEpp: pure C++ implementation of TFHE cryptosystem. <https://github.com/virtualesecureplatform/TFHEpp>. (2020).
- [13] Kotaro Matsuoka, Ryotaro Banno, Naoki Matsumoto, Takashi Sato, and Song Bian. 2021. Virtual Secure Platform: A Five-Stage Pipeline Processor over TFHE. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association. <https://www.usenix.org/conference/usenixsecurity21/presentation/matsuoka>
- [14] Oded Regev. 2005. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing (STOC '05)*. Association for Computing Machinery, New York, NY, USA, 84–93. <https://doi.org/10.1145/1060590.1060603>
- [15] R.L. Rivest, L. Adleman, and M.L. Dertouzos. 1978. On data banks and privacy homomorphisms. In *Foundations on Secure Computation*, Academia Press. 169–179.
- [16] Clifford Wolf. 2013. Yosys Open SYnthesis Suite. <https://yosyshq.net/yosys/>. (2013). Accessed 09/21/2021.