IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 4, pp. 205–230. DOI:10.46586/tches.v2024.i4.205-230

Fast Transciphering Via Batched And Reconfigurable LUT Evaluation

Leonard Schild[1], Aysajan Abidin[1] and Bart Preneel[1]

COSIC KU Leuven, Leuven, Belgium, firstname.lastname@kuleuven.be

Abstract. Fully homomorphic encryption provides a way to perform computations in a privacy preserving manner. However, despite years of optimization, modern methods may still be too computationally expensive for devices limited by speed or memory constraints. A paradigm that may bridge this gap consists of transciphering: as fully homomorphic schemes can perform most computations obliviously, they can also execute the decryption circuit of any conventional block or stream cipher. Hence, less powerful systems may continue to encrypt their data using classical ciphers that may offer hardware support (e.g., AES) and outsourcing the task of transforming the ciphertexts into their homomorphic equivalent to more powerful systems. In this work, we advance transciphering methods that leverage accumulator-based schemes such as Torus-FHE (TFHE) or FHEW. To this end, we propose a novel method to homomorphically evaluate look-up tables in a setting in which encrypted digits are provided on base 2. At a high level, our method relies on the fact that functions with binary range, i.e., mapping values to $\{0, 1\}$, can be evaluated at the same computational cost as negacyclic functions, relying only on the default functionality of accumulator based schemes. To test our algorithm, we implement the AES-128 encryption circuit in OPENFHE and report timings of 67 s for a single block, which is 25% faster than the state of the art and in general, up to 300% faster than other recent works. Furthermore, we achieve this speedup without relying on an instantiation that leverages a power of 2 modulus and can exploit the natural modulo arithmetic of modern processors.

Keywords: Fully Homomorphic Encryption · Transciphering · Lookup Table Evaluation · FHE · LUT · AES

1 Introduction

Unlike traditional encryption methods, fully homomorphic encryption (FHE) enables computations on encrypted data without decryption, thereby preserving the confidentiality of sensitive information throughout the entire computational process. As such, FHE has applications in scenarios where secure outsourcing of computation is imperative, such as in cloud computing environments or collaborative data analysis across distributed networks. However, despite extensive optimization efforts, FHE encryption remains excessively demanding in computation for devices constrained by speed or memory limitations. A potential solution is transciphering, which utilizes FHE's capability to evaluate the decryption circuit of any conventional block- or streamcipher. Consequently, low-end systems can encrypt their data using traditional ciphers, possibly with hardware support (such as AES), while delegating the conversion of ciphertexts into their homomorphic equivalents to the cloud. Although the transciphering of classical ciphers poses little challenges in theory, the actual execution may be rather inefficient with regards to time or memory. This problem led to the creation of stream and block ciphers designed to make transciphering very efficient, such as Chaghri [AMT22] and RASTA [DEG+18]. Such

Received: 2024-04-15 Accepted: 2024-06-15 Published: 2024-09-05

schemes are, in practice, much faster to transcipher while avoiding shortcomings of actual FHE schemes, such as large ciphertext size.

Despite having more beneficial properties than classical ciphers, adoption has been slow for numerous reasons. On the one hand, vendors may prefer to rely on ciphers with a long history of cryptanalysis. Furthermore, many processors provide hardware acceleration for classical ciphers, such as AES-NI for x86, AES instructions through ARM cryptographic extensions and AES-Accelerator on ESP32 devices. Such support allows encryption to be highly efficient even on embedded systems. Finally, transciphering of classical ciphers may be easier to adopt since only methods to generate FHE keys need to be implemented, and existing cryptographic libraries need not be updated, putting fewer burdens on vendors of closed-source systems or other devices in which only a single party can modify software such as smart watches.

1.1 Contribution

Our primary contribution consists of a new homomorphic LUT evaluation algorithm for binary digits that immediately improves the state of the art with regards to AES transciphering. As we rely on accumulator based schemes such as TFHE by Chilloti et al. [CGGI20] we give high level overview of the primitives we use and refer to Sec. 2 for more details.

In our context, we rely on Regev encryptions [Reg09] that are given by learning-with error-samples or LWE samples. Then, for a message $m \in \mathbb{Z}_t, t \in \mathbb{N}$ and ciphertext modulus q, an encryption is given by $\mathrm{LWE}(\frac{q}{t}m) = [\vec{a}^\top, b]^\top$ where $b = \langle \vec{a}, \vec{s} \rangle + \frac{q}{t}m + e \pmod q$ and $\vec{a}$ is a random vector, $\vec{s}$ is the (symmetric) secret key and e is a small error variable. It is easy to see that LWE samples are additively homomorphic by default, but every addition will increase the magnitude of the error part. To reset the error, we rely on the bootstrapping operation.

The starting point of this step is an accumulator or ring-LWE sample (RLWE) $\mathrm{acc} = \mathrm{RLWE}(\mathfrak{p})$ of a polynomial $\mathfrak{p} \in \mathcal{R} := \mathbb{Z}_Q[X]/(X^N + 1)$ with $\log_2(N) \in \mathbb{N}, Q >> q$. For simplicity, we assume that $q = 2N$. RLWE samples are defined in an analogous way to LWE samples, i.e., $\mathrm{RLWE}(\mathfrak{p}) = [\mathfrak{a}, \mathfrak{b}], \mathfrak{b} = \mathfrak{a}\mathfrak{s} + \mathfrak{p} + \mathfrak{e}$ where $\mathfrak{a}, \mathfrak{s}, \mathfrak{e} \in \mathcal{R}$. Then, given such an accumulator acc, encryptions of the coefficients $\vec{s}_i$ of the LWE secret vector, possibly under another encryption scheme, the blind-rotation step which is the major operation of the bootstrapping procedure will compute $\mathrm{RLWE}(\mathfrak{r}) = \mathrm{RLWE}(\mathfrak{p} \cdot X^{b - \langle \vec{a}, \vec{s} \rangle \pmod{2N}})$ for a sample $\mathrm{LWE}(\frac{q}{t}m) = [\vec{a}, b]^\top$. Note that the modular reduction by $2N$ in the exponent stems from the order of the multiplicative subgroup of $\mathcal{R}$ generated by X. More importantly, the error term of the output RLWE sample is *independent* of the noise in the LWE sample. The major strength of this operation is that the constant coefficient $\mathfrak{r}_0$ will be equal to $(-1)^v \cdot \mathfrak{p}_{N-(b-\langle \vec{a}, \vec{s} \rangle) \pmod N}$ with $v = 0$ if $b - \langle \vec{a}, \vec{s} \rangle \pmod{2N} \in \{0, N+1, ..., 2N-1\}$ and $v = 1$ otherwise. As it is possible to derive an LWE sample under modulus q for a single coefficient of an RLWE sample (cf. Sec. 2), the previous observation highlights that it is possible to encode lookup tables into $\mathfrak{p}$ as long as we can predict or ignore the sign flip induced by v. Then, we can make two key observations:

Obs. 1 Any function $f : \{0, 1\} \mapsto \{0, w\} \subset \mathbb{Z}_Q$ can be easily computed with the blind-rotation operation. In this setting, the input is given by an LWE sample $\mathrm{LWE}(\frac{q}{2}m)$ and adding $\frac{q}{4}$ to the b component we can observe that $m = 0 \Rightarrow v = 0$ and $m = 1 \Rightarrow v = 1$ as long as $|e| \leq \frac{q}{4}$. Then, setting every coefficient of $\mathfrak{p}$ to $\bar{w} = -\left\lfloor \frac{w}{2} \right\rceil$ where division by 2 is done over the real numbers, we see that $\mathfrak{r}_0 = (-1)^m \cdot \bar{w}$ after blind-rotation and finally, after deriving an LWE sample we can simply add $\left\lfloor \frac{w}{2} \right\rceil$ to obtain the target values.

Obs. 2 We have previously assumed that $q = 2N$. Another interesting setting is $q = N$, where we can point out that any function $f : \mathbb{Z}_t \mapsto \{0, \frac{Q}{2}\}$ may be evaluated. Note that under modulus $q = N$, we may write $b - \langle \vec{a}, \vec{s} \rangle = \frac{q}{t}m + e + k \cdot N$ and the blind-rotation computes $\text{RLWE}((-1)^k \cdot \mathfrak{p} \cdot X^{\frac{q}{t}m+e \pmod N})$ Then, the sign flip no longer depends on the encrypted message but is rather random and setting the coefficients of $\mathfrak{p}$ to either $\left\lfloor \frac{Q}{2} \right\rceil$ or 0, we can safely ignore the sign flips as

$$\left| \frac{Q}{2} - (-\frac{Q}{2} \pmod Q) \right| \in \{0, 1\},$$

i.e. we incurr a negligible error in the worst case.

We can now present our contributions:

LUT evaluation. We show a new way of evaluating lookup tables indexed by several, binary digits encrypted as LWE samples. More specifically, our input is given by u LWE samples $\text{ct}^i = \text{LWE}(\frac{q}{2}B_i), B_i \in \{0,1\}$ and we wish to evalute a lookup table $f : \{0,1\}^u \mapsto \{0,1\}^u$. We split the LUT evaluation into three stages:

- **Composition:** First, a subset of κ bits is selected. Then, using Observation 1, we rescale the bits such that they can be composed into a unique encrypted integer ct^κ, i.e. the $i - th$ bit B_i is mapped from $\{0,1\}$ to $\{0, 2^i\}$ and all κ outputs are summed.
- **Output Generation:** Using Observation 1, we bootstrap ct^κ so that we compute all possible outputs that conform to the κ packed bits in ct^κ. However, since ct^κ does not incorporate information about the last $u - \kappa$ bits we need to generate outputs for every combination of the remaining bits.
- **Selection:** In the final step, we use the remaining bits to select the correct result from the previously generated outputs. To this end, we rely on Observation 1 and the identity
$$\frac{p+q}{2} + S\left(\frac{p-q}{2}\right) = \begin{cases} p & \text{if } S = 1 \\ q & \text{if } S = -1 \end{cases}$$
by letting the sign flip in the bootstrapping take the place of S.

The most important advantage of our approach is (re)configurability: by splitting the LUT evaluation into multiple stages that are largely independent, we may optimize our parameters for each step taking their requirements into account. For example, each stage will perform blind-rotation with regards to different input plaintext spaces. This has an impact on the magnitude of moduli used and the ring dimension N, both of which will have a direct impact on performance. For the sake of conciseness, we left out some details in the previous overview which will be discussed later on, such as how to generate every output efficiently or how to manage the noise growth of certain operations.

AES Transciphering. At a high level, the key components of our AES transciphering are LWE sample addition, LUT evaluation (Algorithm 8), and homomorphic Galois multiplication (Algorithm 9). To begin with, we encode an AES state $\mathcal{S}$ as a rank 3 tensor of dimension $4 \times 4 \times 8$ containing LWE samples encrypting the individual bits of the state byte, which are encoded as

$$\mathcal{S}_{i,j,k} = \text{LWE}\left(\Delta_{q,2} \cdot [B_{i,j}]_k\right),$$

where $[B_{i,j}]_k$ is the k-th bit of the state byte at entry i, j and $\Delta_{q,2} = \left\lfloor \frac{q}{2} \right\rceil$. This encoding makes XOR operations effectively free as they can be computed through the linear homomorphism of LWE samples. We homomorphically implement the four steps of AES round function as follows.

- AddRoundKey, which XORs the key schedule with the state matrix, requires only LWE sample addition;
- SubBytes, which evaluates a bijection on the entries of the state matrix, is implemented using our LUT evaluation technique;
- ShiftRows, which permutes the entries of the state matrix, is just a permutation of the state tensor;
- MixColumns, which is a linear function over the columns of the state matrix (or $GF(2^8)$), is implemented using LWE sample addition and a homomorphic multiplication algorithm over $GF(2^8)$.

1.2 Related Works

We briefly discuss related works, focusing on TFHE-based lookup table evaluation and transciphering.

By default, accumulator-based schemes such as TFHE by Chilotti et al. [CGGI20] and FHEW by Ducas and Micciancio [DM15] can evaluate arbitrary lookup tables, provided the table fulfills a negacyclity property, i.e. for a value m, $\mathsf{LUT}[m + \frac{q}{2}] = -\mathsf{LUT}[m]$ where q is the ciphertext modulus. This restriction was removed in follow-up works [CLOT21, KS22, LMP23]. Although, in theory, the lookup tables may have an arbitrary input and output size, the performance of the primary bottleneck, bootstrapping, degrades quickly once the input plaintext size exceeds 5-6 bits. To approach this issue, Guimarães et al. [GBA21] introduce a chain- and tree-based method to evaluate lookup tables by splitting inputs into d digits in base B. The chain-based method can efficiently evaluate functions with carry-like properties, such as addition circuits, whereas the tree-based method can evaluate arbitrary lookup tables. The number of bootstrap of the tree base method is $O(B^d)$ by default, but may be reduced to $O(d)$ by leveraging circuit bootstrapping [CGGI20] to transform LWE samples into RGSW samples which were introduced by Gentry et al. [GSW13] and can be viewed as matrices of RLWE sample that are also multiplicatively homomorphic. We note that the use of techniques in [GBA21] is restricted to a setting in which the most significant bit of the phase of the input digits is known.

In the context of transciphering, we focus on several recent works that perform AES-128 transciphering. In [TCBS23], the authors leverage the tree-based method of [GBA21] for every step of the AES circuit, including XOR operations. The authors test several different bases for their digits and obtain a sequential time of 270 s. In 2023, Wei et al. [WWL+23] significantly decrease the transciphering time on a processor of similar speed. In their work, the authors perform operations over bits in which case all operations except the evaluation of the S-BOX of the AES circuit can be performed without bootstrapping. To compute the S-BOX, the authors employ the circuit bootstrapping technique and then evaluate using horizontal and vertical packing techniques from [CGGI20] to obtain a runtime of 87 s. Finally, a novel approach to homomorphically evaluate functions has recently been introduced by Bon et al [BPR23]. Their methodology allows the evaluation of Boolean functions in several variables, but in some circumstances, they only use a single blind-rotation operation. To showcase their algorithm, the authors implement the AES SBOX evaluation and obtain a runtime of 105 seconds. We will revisit the aforementioned works later on in the evaluation.

2 Preliminaries

In this section, we introduce the notation that we use throughout this work. Furthermore, we give a brief introduction to the basic building blocks of TFHE and other related cryptosystems.

2.1 Notation

The letters q and Q will always denote moduli that may be prime and in general q is a power of 2. Then, we denote the integers modulo Q as $\mathbb{Z}_Q$ and the set $\{0, 1\}$ as $\mathbb{B}$. For N a power of two, $\phi_{2N} = (X^N + 1)$ is the $2N$-th cyclotomic polynomial and we define $\mathcal{R}_{N,Q}$ as $\mathbb{Z}_Q[X]/\phi_{2N}(X)$. For vectors $\vec{p}$ or polynomials $\mathfrak{p}$, we denote the i-th coefficient as $\vec{p}_i$ and $\mathfrak{p}_i$, respectively, and when we write $\vec{c}$ or $\mathfrak{c}$ for a constant $c \in \mathbb{Z}$, we mean a vector or polynomial containing the constant c in every coefficient.

Let χ_Q be a distribution over $\mathbb{Z}_Q$, then we call χ_Q $\mathcal{B}$-bounded if its variance is less or equal to $\mathcal{B}$. When referring to a polynomial $\mathfrak{p}$ being distributed according to χ_Q, we assume that all coefficients of $\mathfrak{p}$ were sampled from this distribution. The vector $\vec{s}$ and polynomial $\mathfrak{s}$ will denote secret keys that may be sampled from special distributions such as discrete Gaussian or binary distributions. By $\mathrm{L} \in \mathbb{N}$ we denote a basis inducing a number of digits ℓ depending on context, e.g. $\mathrm{l} = \lceil \log_{\mathrm{L}}(Q) \rceil$. Throughout this paper, $\langle k \rangle = \{0, 1, ..., k-1\}$ and $[x]_i$ will denote the i-th least significant bit of $x \in \mathbb{Z}$. Finally, t will denote a message space and when we operate under a modulus q, $\Delta_{q,t} = \left\lfloor \frac{q}{t} \right\rceil$ is a scaling parameter.

Given a dimensional parameter $n \in \mathbb{N}$, a modulus $q \in \mathbb{N}$ and a $\mathcal{B}$-bounded error distribution χ_q, we define for a value $m \in \mathbb{Z}_q$ and a secret vector $\vec{s} \in \mathbb{Z}_q^n$

$$\mathrm{LWE}_{\vec{s}}(m) : = \left[\vec{a}^\top, b\right]^\top \in \mathbb{Z}_q^n \times \mathbb{Z}_q$$
$$b = \langle \vec{a}, \vec{s} \rangle + m + e, e \xleftarrow{\$} \chi_q,$$

where the coefficients of $\vec{a}$ are sampled uniformly from $\mathbb{Z}_q$. We shall let φ denote the phase of an LWE ciphertext ct, i.e., $\varphi(\mathrm{ct}) = b - \langle \vec{a}, \vec{s} \rangle = m + e$. Similarly, given a polynomial ring $\mathcal{R}_{N,Q}$ and an error distribution χ_Q over $\mathbb{Z}_Q$ we define for a message $\mathfrak{m} \in \mathcal{R}_{N,Q}$

$$\mathrm{RLWE}_{\mathfrak{s}}(\mathfrak{m}) := [\mathfrak{a}, \mathfrak{b}]^\top \in \mathcal{R}_{N,Q}^2 \,,$$

where the coefficients of $\mathfrak{a}$ are sampled uniformly from $\mathbb{Z}_Q$ and $\mathfrak{b} = \mathfrak{a} \cdot \mathfrak{s} + \mathfrak{m} + \mathfrak{e}$, here the coefficients of $\mathfrak{e}$ are sampled from χ_Q. We note that by construction LWE and RLWE samples are both additively homomorphic and homomorphic with regards to scalar multiplication. For the sake of conciseness, we will occasionally drop the subscript for LWE and RLWE samples unless it is necessary, i.e., we write LWE(m) or RLWE($\mathfrak{m}$).

2.2 Bootstrapping in Accumulator Based Schemes

All fully homomorphic encryption schemes encrypt their values by introducing a noise variable whose magnitude will increase with each operation performed. Consequently, it may happen that at some point that a ciphertext becomes impossible to decrypt. The bootstrapping operation, which will vary from scheme to scheme, provides a method that given a noisy ciphertext outputs a new ciphertext, encoding the same value but with a noise magnitude independent of the input noise. In this subsection, we briefly introduce the basic building blocks of bootstrapping in accumulator based schemes.

We begin by noting that it is possible to extract an LWE sample from an RLWE sample without increasing the noise or providing additional key material, as shown in Algorithm 1. In this case, the LWE sample output by the algorithm is encrypted with regards to the coefficients of the secret ring polynomial of the input. Next, we introduce the notion of modulus switching [BV11], described in Algorithm 2. The modulus switching operation transforms an LWE sample under modulus Q and dimension n, containing an encoding of a message m, into a new LWE sample under modulus q encoding the same message with a different noise variance. The magnitude of that noise depends on the ratio $\frac{Q}{q}$, the coefficient distribution of the secret key and the way rounding is performed. For more details, we refer to [LMK+23].

Algorithm 1: SampleExtract

```
Input: An RLWE sample acc = $\mathrm{RLWE}_{\mathfrak{s}}(\mathfrak{m}) = [\mathfrak{a}, \mathfrak{b}]^\top \in \mathcal{R}^2_{N,Q}$
Input: An index $0 \le k < N$
Output: An LWE sample ct = $\mathrm{LWE}_{\vec{s}}(\mathfrak{m}_i)$ where $\vec{s}_i = \mathfrak{s}_i$
$b \leftarrow \mathfrak{b}_i$;
for $i = 0$ to $N - 1$ do
    $\vec{a}_i \leftarrow \begin{cases} -\mathfrak{a}_{k-i+N} & k - i < 0 \\ \mathfrak{a}_{k-i} & k - i \ge 0 \end{cases}$
end
return ct = $[\vec{a}^\top, b]^\top$
```

Algorithm 2: ModSwitch$_{Q \to q}$

```
Input: An LWE sample $\mathrm{ct}_Q = \mathrm{LWE}(\Delta_{Q,t} m) = [\vec{a}^\top, b]^\top \in \mathbb{Z}_Q^n \times \mathbb{Z}_Q$
Output: An LWE sample $\mathrm{ct}_q = \mathrm{LWE}(\Delta_{q,t} m) = [\vec{c}^\top, d]^\top \in \mathbb{Z}_q^n \times \mathbb{Z}_q$
$d \leftarrow \left\lfloor \frac{q \cdot b}{Q} \right\rceil$;
for $i = 0$ to $n - 1$ do
    $\vec{c}_i \leftarrow \left\lfloor \frac{q \cdot \vec{a}_i}{Q} \right\rceil$;
end
return $\mathrm{ct}_q = [\vec{c}^\top, d]^\top$
```

We now come to the key switching procedure [BV11], described in Algorithm 3. Given an LWE sample under a secret key $\vec{s}$ and a *key switching key* as input, it outputs a new LWE sample encoding the same message under a different secret key $\vec{S}$. We note that the key switching operation can easily be extended to output an RLWE sample by letting the key switching key in Algorithm 3 be a tensor of RLWE samples instead of LWE samples and in these cases we denote the key-switching by RLWEKeySwitch$_{\vec{s} \to \mathfrak{s}}$.

Algorithm 3: KeySwitch$_{\vec{s} \to \vec{S}}$

```
Input: An LWE sample ct = $\mathrm{LWE}_{\vec{s}}(\tilde{m}) = [\vec{a}^\top, b]^\top \in \mathbb{Z}_q^n \times \mathbb{Z}_q$
Input: A key switching key KsK of dimension $n \times \mathrm{l} \times \mathrm{L}$ where
       $\mathrm{KsK}_{i,j,k} = \mathrm{LWE}_{\vec{S}}\left(k \cdot L^j \cdot \vec{s}_i\right) \in \mathbb{Z}_q^{n'} \times \mathbb{Z}_q$
Output: An LWE sample $\mathrm{ct}_{\mathsf{out}} = [c^\top, d]^\top \in \mathrm{LWE}_{\vec{S}}$
Set $\mathrm{ct}_{\mathsf{out}} \leftarrow [\vec{0}^\top, b]^\top$;
for $i = 0$ to $n - 1$ do
    for $j = 0$ *to* to $l - 1$ do
        Set $a_{i,j}$ to be the $j$-th digit of $\vec{a}_i$ in base $L$;
        Compute $\mathrm{ct}_{\mathsf{out}} \leftarrow \mathrm{ct}_{\mathsf{out}} - \mathrm{KsK}_{i,j,a_{i,j}}$;
    end
end
return $\mathrm{ct}_{\mathsf{out}}$
```

The final building block is the blind-rotation operation BlindRotate. We shall not describe this procedure in detail since there is a variety of techniques to perform this step [CGGI20],[DM15],[LMK+23], each with different trade-offs. However, we do note that the general approach expects an LWE sample ct = $\mathrm{LWE}(\tilde{m})$ usually under modulus $2N$ and a possibly noiseless RLWE sample or accumulator acc = $\mathrm{RLWE}(\mathfrak{v})$ with $\mathfrak{v}(X) \in \mathcal{R}_{N,Q}$.

Then, it holds that

$$\mathsf{BlindRotate}(\mathrm{ct}, \mathrm{acc}) = \mathrm{RLWE}_{\mathfrak{s}}(\mathfrak{v} \cdot X^{\varphi(\mathrm{ct}) \pmod{2N}}), \tag{1}$$

where the magnitude of the noise component of the output RLWE sample is *independent* of ct. Since N was chosen as a power of two, $X^k \equiv -X^{k-N}$ for any monomial $X^k \in \mathcal{R}_{N,Q}, N \leq k < 2N$. We will heavily rely on this property and, therefore, give another way of expressing Eqn 1 below in Eqn 2:

$$\mathsf{BlindRotate}(\mathrm{ct}, \mathrm{acc}) = \begin{cases} \mathrm{RLWE}_{\mathfrak{s}}\big(\mathfrak{v} \cdot X^{\varphi(\mathrm{ct}) \pmod{N}}\big) & \text{if } \varphi(\mathrm{ct}) \pmod{2N} < N, \\ \mathrm{RLWE}_{\mathfrak{s}}\big(-\mathfrak{v} \cdot X^{\varphi(\mathrm{ct}) \pmod{N}}\big) & \text{if } \varphi(\mathrm{ct}) \pmod{2N} \geq N. \end{cases} \tag{2}$$

The previously described building blocks can be composed in different ways to create a full bootstrapping procedure, which decreases the noise of an input LWE sample and can also be leveraged to evaluate an arbitrary function at the same time. For a thorough treatment, we refer to any of [CGGI20, DM15, LMK$^+$23, KS22].

Finally, we stress that all major blind-rotation approaches [CGGI20, DM15, LMK$^+$23] induce noise additively and do not amplify the noise variance of the input RLWE sample beyond that. To show this in case of the CGGI/TFHE blind-rotation, we refer to [KS22], but note in general that this fact can be reduced to the observation that products between RLWE and RGSW samples [GSW13] increase the noise of the RLWE sample in proportion to the message of the RGSW sample, which in the context of blind-rotation usually consists of monomials, i.e., the error is not amplified.

3 Reconfigurable LUT evaluation

This section shows a specific approach evaluating lookup tables (LUTs) in settings where both the input and output are given as bits. More specifically, we are given u LWE samples $(\mathrm{ct}_q^i)_{i \in \langle u \rangle}$ encoding bits, i.e., $\mathrm{ct}_q^i = \mathrm{LWE}\,(\Delta_{q,2} B_i)$, that correspond to an integer $M = \sum_{i=0}^{u-1} B_i \cdot 2^i$. Then, given a map $f : \mathbb{Z}_{2^u} \mapsto \mathbb{Z}_{2^u}$ we aim to obtain each bit $\beta_i = [f(M)]_i, i \in \langle u \rangle$ without computing $f(M)$. In other words, we immediately compute

$$\mathrm{ct}_{\mathsf{out}}^i = \mathrm{LWE}\,(\Delta_{Q,2}\beta_i)\,.$$

with Q possibly equal to q. We proceed as follows: in Sect. 3.1 we illustrate some properties that facilitate the evaluation of binary functions via bootstrapping and describe methods to perform approximate multiplexing of RLWE ciphertext in Sect. 3.2 in order to finally construct the LUT evaluation technique in Sect. 3.3 .

3.1 Functional Binary Bootstrapping

First, let $\mathrm{ct} = \mathrm{LWE}(\Delta_{q,t} m)$ be an encoding of a message $m \in \mathbb{Z}_t$. Then, any function $f : \mathbb{Z}_t \mapsto \mathbb{B}^u, u > 0$, can be evaluated in a single blind-rotation. Generally, the blind-rotation step assumes the input to be under modulus $2N$, but the input may also be provided under modulus N. Recall that the multiplicative order of $\mathcal{R}_{N,Q}$ is $2N$ and that for an LWE sample $\mathrm{ct} = \mathrm{LWE}_{\vec{s}}(m)$ under modulus N, we can write

$$\varphi(\mathrm{ct}) = \langle \vec{a}, \vec{s} \rangle + m + e + k \cdot N\,, \tag{3}$$

with $\langle \vec{a}, \vec{s} \rangle + m + e < N$. Then, if we supply the same LWE sample modulo N to the blind-rotation procedure, we can observe that in Eqn. 2 the sign flip now purely depends on whether k is even or odd in Eqn. 3. Furthermore, the coefficients of the polynomial in the accumulator must contain multiples of $\left\lfloor \frac{Q}{2} \right\rceil$ in order to conform to the binary output space. Then, the following equation

$$\left| \left\lfloor \frac{Q}{2} \right\rceil - \left(- \left\lfloor \frac{Q}{2} \right\rceil \pmod{Q} \right) \right| \in \{0, 1\} \tag{4}$$

states that sign flips with such a scaling factor can effectively be neglected and together with the previous discussion induces a method to evaluate any function with one or several output bits using only in a single blind-rotation. We give the concrete steps in Algorithm 4 and ignore the parameter C' for now. We note that the algorithm outputs the bits as LWE samples modulo Q and under a secret key whose coefficients correspond to the coefficients of the RLWE key. If a different output key (or modulus) is required, we may use the key (or modulus) switching from Sect. 2. Lemma 1 states the correctness of Algorithm 4.

```
Algorithm 4: EvalBinFunc
  Input: $\mathrm{ct}_q = \mathrm{LWE}_{\vec{s}}(\Delta_{q,t} m) = [\vec{a}^\top, b]^\top$
  Input: A map $f : \mathbb{Z}_t \mapsto \mathbb{B}^u$ with $f(x) = [f_0(x), ..., f_{u-1}(x)]$
  Input: Optional Plaintext space $C'$
  Output: $\mathrm{ct}^i_{\mathsf{out}} = \mathrm{LWE}_{\vec{S}}(\Delta_{Q,2}\beta_i)$ with $\beta_i = f_i(m)$
  $(\mathfrak{p}^i)_{i \in \langle u \rangle} \leftarrow \mathfrak{o} \in \mathcal{R}_{N,Q}$;
  for $j = 0$ to $t - 1$ do
      for $i = 0$ to $u - 1$ do
          $\mathfrak{z}^i_{j \cdot \frac{N}{t}} \leftarrow f_i(t - j)$;
      end
  end
  for $i = 0$ to $u - 1$ do
      $\epsilon_i \leftarrow [||\mathfrak{z}^i||_0 \geq \frac{t}{2}]$;
      for $j = 0$ to $t - 1$ do
          $\mathfrak{p}^i_{j \cdot \frac{N}{t}} \leftarrow \epsilon_i + (-1)^{\epsilon_i} \cdot \mathfrak{z}^i_{j \cdot \frac{N}{t}}$;
      end
  end
  $C \leftarrow 2$ if $C'$ is not provided, else $C'$;
  $\mathrm{ct}_N \leftarrow \mathsf{ModSwitch}_{q \to N}(\mathrm{ct}_q)$;
  $\mathrm{acc} \leftarrow \left[\mathfrak{o}, \left\lfloor \frac{Q}{C} \right\rceil \cdot X^{-\frac{N}{2t}} \cdot \sum_{i=0}^{\frac{N}{t}} X^i \right]$;
  $\mathrm{acc}_{\mathsf{out}} \leftarrow \mathsf{BlindRotate}(\mathrm{ct}_N, \mathrm{acc})$;
  for $i = 0$ to $u - 1$ do
      $\mathrm{acc}^i \leftarrow \mathrm{acc}_{\mathsf{out}} \cdot \mathfrak{p}^i$;
      $\mathrm{ct}^i_{\mathsf{out}} \leftarrow \mathsf{SampleExtract}(\mathrm{acc}^i, 0)$;
      if $\epsilon_i = 1$ then
          $\mathrm{ct}^i_{\mathsf{out}} \leftarrow \left[\vec{0}, \left\lfloor \frac{Q}{C} \right\rceil\right]^\top - \mathrm{ct}^i_{\mathsf{out}}$;
      end
  end
  return $[\mathrm{ct}^i_{\mathsf{out}}]_{i \in \langle u \rangle}$
```

Lemma 1. *Let* $\mathrm{ct}_q = \mathrm{LWE}_{\vec{s}}(\Delta_{q,t} m) = [\vec{a}^\top, b = \langle \vec{a}, \vec{s} \rangle + \Delta_{q,t} m + e]^\top$, $f : \mathbb{Z}_t \mapsto \mathbb{B}^u$ *with* $f(x) = [f_0(x), ..., f_{u-1}(x)]$ *and* $C = C' = 2$. *Then, if* $|e| < \frac{q}{2t}$, *Algorithm 4 outputs* $[\mathrm{ct}^i_{\mathsf{out}}]_{i \in \langle u \rangle}$ *such that* $\mathrm{ct}^i_{\mathsf{out}} = \mathrm{LWE}_{\vec{S}}(\Delta_{Q,2}\beta_i)$ *with* $\beta_i = f_i(m)$ *and noise variance bounded by*

$$\mathcal{B}_{\mathsf{out}} \leq \frac{t}{2} \mathcal{B}_{BR} \,.$$

Proof. First, we note that for all $i \in \langle u \rangle, j \in \langle t \rangle$ it holds that

$$\mathfrak{p}^i_{j\cdot\frac{N}{t}} = \begin{cases} f_i(t-j) & \text{if } ||\mathfrak{z}||_0 < \frac{t}{2} \Leftrightarrow \epsilon_i = 0\,, \\ 1 - f_i(t-j) & \text{else}\,. \end{cases}$$

Then, we have that

$$\begin{aligned}
\varphi(\mathrm{acc}^i) &= \varphi(\mathfrak{p}^i \cdot \mathrm{acc}) \\
&= \mathfrak{p}^i \cdot X^{\varphi(\mathrm{ct}_N)} \cdot \left\lfloor \frac{Q}{2} \right\rceil \cdot X^{-\frac{N}{2t}} \cdot \sum_{i=0}^{\frac{N}{t}} X^i + \mathfrak{p}^i \cdot \mathfrak{e} \\
&= X^{\frac{N}{t}m + e_N} \cdot \left\lfloor \frac{Q}{2} \right\rceil \cdot X^{-\frac{N}{2t}} \cdot \mathfrak{p}^i \cdot \sum_{i=0}^{\frac{N}{t}} X^i + \mathfrak{p}^i \cdot \mathfrak{e} \\
&= X^{\frac{N}{t}m + e_N} \cdot \left\lfloor \frac{Q}{2} \right\rceil \cdot X^{-\frac{N}{2t}} \cdot \sum_{j=0}^{t-1} \sum_{i=0}^{\frac{N}{t}} \mathfrak{p}^i_{j\cdot\frac{N}{t}} \cdot X^i + \mathfrak{p}^i \cdot \mathfrak{e} \\
\implies \varphi(\mathrm{acc}^i)_0 &= \left(X^{\frac{N}{t}m + e_N} \cdot \left\lfloor \frac{Q}{2} \right\rceil \cdot X^{-\frac{N}{2t}} \cdot \sum_{j=0}^{t-1} \sum_{i=0}^{\frac{N}{t}} \mathfrak{p}^i_{j\cdot\frac{N}{t}} \cdot X^i \right)_0 + (\mathfrak{p}^i \cdot \mathfrak{e})_0 \\
&= \left\lfloor \frac{Q}{2} \right\rceil \cdot \left(\sum_{j=0}^{t-1} \sum_{i=0}^{\frac{N}{t}} \mathfrak{p}^i_{j\cdot\frac{N}{t}} \cdot X^i \right)_{\frac{N}{t}(t-m) - e_N - \frac{N}{2t} \pmod N} + (\mathfrak{p}^i \cdot \mathfrak{e})_0 \\
&= \left\lfloor \frac{Q}{2} \right\rceil \cdot \mathfrak{p}^i_{(t-m)\cdot\frac{N}{t}} + (\mathfrak{p}^i \cdot \mathfrak{e})_0\,,
\end{aligned}$$

where the last step follows from the assumption that $|e| \leq \frac{q}{2t} \Leftrightarrow |e_N| \leq \frac{N}{2t}$ and where $\mathfrak{e}$ is the error polynomial induced by blind-rotation. Hence:

$$\begin{aligned}
\varphi(\mathrm{ct}^i_{\mathsf{out}}) &= \begin{cases} \varphi(\mathrm{acc}^i)_0 & \text{if } \epsilon_i = 0 \\ \left\lfloor \frac{Q}{2} \right\rceil - \varphi(\mathrm{acc}^i)_0 & \text{if } \epsilon_i = 1 \end{cases} \\
&= \begin{cases} \left\lfloor \frac{Q}{2} \right\rceil \cdot \mathfrak{p}^i_{(t-m)\cdot\frac{N}{t}} + (\mathfrak{e} \cdot \mathfrak{p}^i)_0 & \text{if } \epsilon_i = 0 \\ \left\lfloor \frac{Q}{2} \right\rceil - \left\lfloor \frac{Q}{2} \right\rceil \cdot \mathfrak{p}^i_{(t-m)\cdot\frac{N}{t}} - (\mathfrak{e} \cdot \mathfrak{p}^i)_0 & \text{if } \epsilon_i = 1 \end{cases} \\
&= \left\lfloor \frac{Q}{2} \right\rceil \cdot f_i(m) \pm (\mathfrak{e} \cdot \mathfrak{p}^i)_0\,.
\end{aligned}$$

Finally, we note that

$$\begin{aligned}
\mathsf{Var}((\mathfrak{e} \cdot \mathfrak{p}^i)_0) = \mathsf{Var}(\mathfrak{e} \cdot \mathfrak{p}^i) &= \mathsf{Var}\left(\sum_{j:\mathfrak{p}^i_j = 1} \mathfrak{e} \cdot X^j \right) \\
&= ||\mathfrak{p}^i||_0 \cdot \mathsf{Var}(\mathfrak{e}) \\
&\leq \frac{t}{2} \mathsf{Var}(\mathfrak{e}) \\
&= \frac{t}{2} \mathcal{B}_{BR}\,.
\end{aligned}$$

□

At a high level, the method works as follows. Initially, a set of binary polynomials encodes the respective output bits. Then, an accumulator is created which can be intuitively seen as a constant with padding and where the number of nonzero entries corresponds to the chunk size of the messages, i.e., $\frac{N}{t}$. Next, we perform a blind-rotation after switching the modulus of the input ciphertext to N. As previously discussed, we can ignore the random sign flip as we used a scaling factor of $\left\lfloor \frac{Q}{2} \right\rceil$. Finally, we multiply the output of the blind-rotation with the polynomials we previously constructed, resulting in an RLWE sample encoding a polynomial whose constant coefficient purely depends on the phase of the input LWE sample.

As we have just seen how to evaluate binary functions and, by extension, bit-decompositions, we may also look at the inverse operation bit composition or de-multiplexing. Composing several bits into an integer can easily be achieved using the default functionality of accumulator-based schemes. The concrete steps are given in Algorithm 5 and leverage the fact that we encode the binary plaintext in the most significant bit of the phase. Initially,

Algorithm 5: AssembleInteger

```
Input: $(\mathrm{ct}_q^i)_{i \in \langle u \rangle} = \mathrm{LWE}\left(\frac{q}{2} B_i\right)$
Input: Output plaintext space $t$
Output: $\mathrm{ct}_{\mathsf{out}} = \mathrm{LWE}\left(\left\lfloor \frac{Q}{t} \right\rceil \cdot \sum_{i=0}^{u-1} 2^i B_i\right)$
$\mathrm{ct}_{\mathsf{out}} = [\vec{0}, 0] \in \mathbb{Z}_Q^N \times \mathbb{Z}_Q$;
$\mathfrak{h} \leftarrow \left\lfloor \frac{Q}{2t} \right\rceil \cdot (1 - X^{\frac{N}{2}}) \left(\sum_{i=0}^{\frac{N}{2}-1} X^i\right)$;
$\mathrm{acc} \leftarrow [\mathfrak{o}, \mathfrak{h}]^\top$;
for $i \in \langle u \rangle$ do
    $\mathrm{ct}_{2N}^i \leftarrow \mathsf{ModSwitch}_{q \to 2N}(\mathrm{ct}_q^i + \left[\vec{0}^\top, \frac{q}{4}\right]^\top)$;
    $\mathrm{acc}^i \leftarrow \mathsf{BlindRotate}(\mathrm{ct}_{2N}^i, 2^i \cdot \mathrm{acc})$;
    $\mathrm{ct}_Q^i \leftarrow \mathsf{SampleExtract}(\mathrm{acc}^i, 0)$;
    $\mathrm{ct}_Q^i \leftarrow \mathrm{ct}_Q^i + \left[\vec{0}, 2^i \cdot \left\lfloor \frac{Q}{2t} \right\rceil\right]$;
    $\mathrm{ct}_{\mathsf{out}} \leftarrow \mathrm{ct}_{\mathsf{out}} + \mathrm{ct}_Q^i$;
end
return $\mathrm{ct}_{\mathrm{out}}$
```

we add $\frac{q}{4}$ to the LWE samples before blind-rotation in order to center the distribution of the phase to $\frac{q}{4} + \frac{q}{2} B_i$, guaranteeing that the most significant bit of the phase matches the bit encoded in the LWE sample. Then, each LWE sample is modulo switched to 2N and blindrotated with an accumulator that guarantees that the sign of the constant term depends on the most significant bit of the input. Afterwards, an LWE sample is extracted and the message is mapped to 0 or a power of two, and finally, all LWE samples are added. For the output variance, we give the following Lemma.

Lemma 2. *Let $\mathcal{B}_{BR}$ be the noise variance induced by the blind-rotation algorithm. Then, Algorithm 5 outputs an LWE sample of dimension N and modulus Q with noise variance $\mathcal{B}_{\mathsf{out}} \leq u \cdot \mathcal{B}_{BR}$.*

Proof. Let $\mathcal{B}_{BR}$ be the noise variance induced by the blind-rotation algorithm. Then all

ct_0^i have an error component with variance equal to $\mathcal{B}_{BR}$. It follows that

$$\begin{aligned}\mathsf{Var}\left(\varphi(\mathrm{ct}_{\mathsf{out}})\right) &= \mathsf{Var}\left(\varphi\left(\sum_{i\in\langle u\rangle}\mathrm{ct}_Q^i\right)\right)\\ &= \sum_{i\in\langle u\rangle}\mathsf{Var}\left(\varphi(\mathrm{ct}_Q^i)\right)\\ &\leq u\cdot\mathcal{B}_{BR}\,.\end{aligned}$$

□

Leveraging the last two algorithms, we can point out that by first applying Algorithm 4 followed by Algorithm 5 we achieve a functional bootstrap without any assumption on the most significant of the original integer input. Compared to recent works like [LMP23],[CLOT21] or [KS22] the number of blind-rotations of the outlined procedure may be higher. However, it may still be preferable in a scenario in which part of the input is given as bits and one input as integer, and the output is to be given as an integer, e.g. when we need to compute a bitwise XOR operation $x \oplus y$ and the right operand is given as bits. Then, inputs do not need to be combined to evaluate this operation via bootstrapping. Furthermore, splitting up the algorithm this way can be favourable as both stages have different requirements with regards to the the parameter sets of blind-rotation. Algorithm 4 needs to support a u_0-bit input space and a binary output space, whereas Algorithm 5 needs to support a binary input space and a u_1-bit output space.

By inverting the order of application, we can similarly construct a binary LUT evaluation technique by first applying Algorithm 5 followed by Algorithm 4, using either u or blind-rotations. For a binary setting, this is already more favourable than the tree-based method given by Guimarães et al. [GBA21], which can have exponential runtime if no scheme switching is employed. Furthermore, we would no longer have to make any assumptions with regards to the most significant bit of the phase of the LWE samples as is required in [CIM19] which is leveraged by [GBA21].

3.2 Approximate Multiplexing

In this subsection, we discuss two techniques allowing us to approximately multiplex between two RLWE samples given an LWE sample encoding a bit as control signal. More formally, given $\mathrm{acc}_0 = \mathrm{RLWE}(\mathfrak{p}), \mathrm{acc}_1 = \mathrm{RLWE}(\mathfrak{q})$, and $\mathrm{ct} = \mathrm{LWE}(\Delta_{q,2}B), B \in \mathbb{B}$ we wish to compute:

$$\mathsf{MUX}(\mathrm{ct}, \mathrm{acc}_0, \mathrm{acc}_1) = \begin{cases}\mathrm{RLWE}(\mathfrak{r}\cdot\mathfrak{p}) & B=0\,,\\ \mathrm{RLWE}(\mathfrak{r}\cdot\mathfrak{q}) & B=1\,,\end{cases} \tag{5}$$

where $\mathfrak{r} \in \mathcal{R}_{N,Q}$ is a polynomial of predictable degree or coefficient norm. The most well known solution to this problem for $\mathfrak{r} = 1$, i.e., the exact setting, is induced by the CMUX gate introduced in [CGGI20]. Then, we can first scheme switch ct to an RGSW sample and apply the CMUX gate afterwards.

However, in many instances we do not need an exact evaluation. More specifically, we will assume that the coefficients of $\mathfrak{p}$ (resp. $\mathfrak{q}$) are distributed in chunks, i.e.,

$$\exists k|N : \mathfrak{p} = \sum_{i=0}^{k-1}\bar{\mathfrak{p}}_i\sum_{j=0}^{\frac{N}{k}-1}X^{\frac{N}{k}\cdot i+j}\,. \tag{6}$$

In this case, we will call the polynomial or corresponding RLWE sample k-redundant. Note that this implies that there are at most k unique coefficients in the polynomial.

Let us consider a setting in which $\deg \mathfrak{p}, \deg \mathfrak{q} < \frac{N}{2}$ and where $\mathrm{acc}_0, \mathrm{acc}_1$ are k-redundant. Note that this implies that the upper $\frac{N}{2}$ coefficients are 0 and effectively only $\frac{k}{2}$ values are contained in the polynomials. Then, let us assume that the error of the LWE sample is bounded by $\frac{q}{2k}$, i.e.,

$$|\varphi(\mathrm{ct}) - \Delta_{q,2}B \pmod q| \leq \frac{q}{2k} \,. \tag{7}$$

Then, a first version of a MUX gate is given by Algorithm 6. At first, acc_0 and acc_1 are combined into a single RLWE sample. Then, the ciphertext is modulo switched to N and therefore a blind-rotation rotates the coefficients of the accumulator by its phase. Post blind-rotation, we extract fixed coefficients which should be equal to the unique coefficients of $\mathfrak{p}$ if $B = 0$ and the coefficients of $\mathfrak{q}$ if $B = 1$, assuming that Eqn. 7 holds. Note that the coefficients may be negated due to the negacyclicty property of the ring. Hence, for $\mathsf{SIG-MUX}$, it holds that $\mathfrak{r} = \pm 1$. We note that in general the values of N and k are

Algorithm 6: $\mathsf{SIG-MUX}$

```
Input: $\mathrm{acc}_0 = \mathrm{RLWE}(\mathfrak{p}), \mathrm{acc}_1 = \mathrm{RLWE}(\mathfrak{q})$
Input: $\mathrm{ct}_q = \mathrm{LWE}(\Delta_{q,2}B)$
Output: $\mathrm{RLWE}(\mathfrak{r}\mathfrak{v})$
$\mathrm{ct}_N \leftarrow \mathsf{ModSwitch}_{q\to N}(\mathrm{ct}_q)$;
$\mathrm{acc} \leftarrow \mathrm{acc}_0 + X^{\frac{N}{2}}\mathrm{acc}_1$;
$\mathrm{acc} \leftarrow X^{N-\frac{N}{4k}} \cdot \mathrm{acc}$;
$\mathrm{acc}_{BR} \leftarrow \mathsf{BlindRotate}(\mathrm{ct}, \mathrm{acc})$;
$\mathrm{acc}_{\mathsf{out}} \leftarrow [\mathfrak{o}, \mathfrak{o}]^\top$;
$\mathfrak{h} \leftarrow \sum_{j=0}^{\frac{N}{2k}-1} X^j$;
for $i = 0$ to $k-1$ do
    $\mathrm{ct}_Q^i \leftarrow \mathsf{SampleExtract}(\mathrm{acc}_{BR}, i \cdot \frac{N}{2k})$;
    $\mathrm{acc}^i \leftarrow \mathsf{RLWEKeySwitch}_{\vec{S}\to\mathfrak{s}}(\mathrm{ct}_Q^i)$;
    $\mathrm{acc}_{\mathsf{out}} \leftarrow \mathrm{acc}_{\mathsf{out}} + X^{\frac{N}{2k}i} \cdot \mathfrak{h} \cdot \mathrm{acc}^i$;
end
return $\mathrm{acc}_{\mathsf{out}}$
```

known ahead of time. Hence, the multiplication by $\mathfrak{h}$ in Algorithm 6 does not need to be explicitly computed, provided that we modify the RLWE key-switching key to include $\mathfrak{h}$, i.e. for an RLWE key-switching key from $\vec{S}$ to $\mathfrak{s}$

$$\mathrm{KsK}_{i,j,k} = \mathrm{RLWE}_{\mathfrak{s}}(\mathfrak{h} \cdot k \cdot L^j \cdot \vec{S}_i) \,.$$

Then, the following lemma holds

Lemma 3. *Let $\mathcal{B}_0, \mathcal{B}_1$ be the noise variance of* $\mathrm{acc}_0, \mathrm{acc}_1$ *and let $\mathcal{B}_{BR}$ be the noise variance induced by the blind-rotation algorithm. Furthermore, let $\mathcal{B}_{KS}$ be the noise variance induced by RLWE key-switching. Then, Algorithm 6 outputs an RLWE sample of noise variance*

$$\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_0 + \mathcal{B}_1 + \mathcal{B}_{BR} + k \cdot \mathcal{B}_{KS} \,.$$

Proof. Let $\mathcal{B}_0, \mathcal{B}_1$ be the noise variance of $\mathrm{acc}_0, \mathrm{acc}_1$ and let $\mathcal{B}_{BR}$ be the noise variance induced by the blind-rotation algorithm. Furthermore, let $\mathcal{B}_{KS}$ be the noise variance induced by RLWE key-switching. We assume that $\mathfrak{h}$ is known ahead of time. Then, we have that

$$\varphi(\mathrm{acc}_{\mathsf{out}}) = \sum_{i=0}^{k-1} (\mathfrak{h} \cdot \varphi(\mathrm{acc}_{BR})_{i\cdot\frac{N}{2k}} + \mathfrak{e}_{KS}^i) \,,$$

where the coefficients of $\mathfrak{e}^i_{KS}$ have a variance of $\mathcal{B}_{KS}$ and the error of acc_{BR} is the sum of the noise of $\mathrm{acc}_0, \mathrm{acc}_1$ and the noise induced by blind-rotation, hence

$$\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_0 + \mathcal{B}_1 + \mathcal{B}_{BR} + k \cdot \mathcal{B}_{KS}. \qquad \square$$

The previously described MUX gate is correct and can be useful in settings where sign flips can be neglected, e.g. when $\forall i : \mathfrak{p}_i \in \{0, \left\lfloor \frac{Q}{2} \right\rceil\}$. However, the method is far from optimal. We are forced to pack every coefficient into a single RLWE sample. Furthermore, since we are only modulo switching to N, we are "wasting" half the order of the ring. Let $S \in \{-1, 1\}$ and consider the identity in Eqn. 8.

$$\mathfrak{p} + \mathfrak{q} + S \cdot (\mathfrak{p} - \mathfrak{q}) = \begin{cases} 2\mathfrak{p} & S = 1, \\ 2\mathfrak{q} & S = -1. \end{cases} \tag{8}$$

There, we see how to select between two polynomials without a need to concatenate the coefficients. Furthermore, it can be easily applied to our context. This time we shall assume $\deg \mathfrak{p}, \deg \mathfrak{q} < N$ that they are k redundant and that Eqn. 7 still holds. Then, we can apply Eqn. 8 homomorphically by modulus switching the ciphertext to 2N and letting the blind-rotation take the role of S, as the signflip it induces depends on the encoded bit. The full specification is given in Algorithm 7, and in this setting, we have that $\mathfrak{r} = 2$, which can be counteracted by pre-multiplying $\mathfrak{p}, \mathfrak{q}$ by $2^{-1} \pmod{Q}$.

Algorithm 7: TWO − MUX

```
Input: $\mathrm{acc}_0 = \mathrm{RLWE}(\mathfrak{p}), \mathrm{acc}_1 = \mathrm{RLWE}(\mathfrak{q})$ where $\mathfrak{p}, \mathfrak{q}$ are $k$ redundant
Input: $\mathrm{ct}_q = \mathrm{LWE}(\Delta_{q,2}B)$
Output: $\mathrm{RLWE}(\mathfrak{r} \cdot \mathfrak{v})$
$\mathrm{ct}_{2N} \leftarrow \mathsf{ModSwitch}_{q \to 2N}(\mathrm{ct}_q + [\vec{0}, \frac{q}{4}])$;
/* Multiplication by $-X^{\frac{N}{2}}$ to counteract addition of $\frac{q}{4}$ */
$\mathrm{acc} \leftarrow -X^{\frac{N}{2}} \cdot X^{N - \frac{N}{2k}} \cdot (\mathrm{acc}_0 - \mathrm{acc}_1)$;
$\mathrm{acc}_{BR} \leftarrow \mathsf{BlindRotate}(\mathrm{ct}_{2N}, \mathrm{acc})$;
$\mathrm{acc}_- \leftarrow [\mathfrak{o}, \mathfrak{o}]^\top$;
$\mathfrak{h} \leftarrow \sum_{j=0}^{\frac{N}{k}-1} X^j$;
for $i = 0$ to $k - 1$ do
    $\mathrm{ct}^i_Q \leftarrow \mathsf{SampleExtract}(\mathrm{acc}_{BR}, i \cdot \frac{N}{k})$;
    $\mathrm{acc}^i \leftarrow \mathsf{RLWEKeySwitch}_{\vec{S} \to \mathfrak{s}}(\mathrm{ct}^i_Q)$;
    $\mathrm{acc}_- \leftarrow \mathrm{acc}_- + X^{\frac{N}{k} i} \cdot \mathfrak{h} \cdot \mathrm{acc}^i$;
end
$\mathrm{acc}_{\mathsf{out}} = \mathrm{acc}_0 + \mathrm{acc}_1 + \mathrm{acc}_-$;
return $\mathrm{acc}_{\mathsf{out}}$
```

Lemma 4. *Let $\mathcal{B}_0, \mathcal{B}_1$ be the noise variance of $\mathrm{acc}_0, \mathrm{acc}_1$ and let $\mathcal{B}_{BR}$ be the noise variance induced by the blind-rotation algorithm. Furthermore, let $\mathcal{B}_{KS}$ be the noise variance induced by RLWE key-switching and let $\mathrm{ct}_q = \mathrm{LWE}(\Delta_{q,2}B)$. Then assuming Eqn. 7 holds and $\mathfrak{h}$ is known ahead of time, Algorithm 7 outputs an RLWE sample of noise variance*

$$\mathcal{B}_{\mathsf{out}} \leq 2 \cdot (\mathcal{B}_0 + \mathcal{B}_1) + \mathcal{B}_{BR} + k \cdot \mathcal{B}_{KS}\,.$$

Proof. Let us assume Eqn. 7 holds, that $\mathfrak{p}, \mathfrak{q}$ are k-redundant and that $\mathfrak{h}$ is known ahead of time. Furthermore, let $\varphi(\mathrm{acc}_0) = \mathfrak{p} + \mathfrak{e}^0$ and $\varphi(\mathrm{acc}_1) = \mathfrak{q} + \mathfrak{e}^1$, where $\mathfrak{e}^0, \mathfrak{e}^1$ are error polynomials variance $\mathcal{B}_0, \mathcal{B}_1$ respectively. Then assuming $\bar{e} = \varphi(\mathrm{ct}_{2N}) - \Delta_{2N,2}B \pmod{2N}$ it holds that for any $i < N$

$$\begin{aligned}\varphi(\mathrm{acc}_{\mathsf{out}})_i &= \varphi(\mathrm{acc}_0)_i + \varphi(\mathrm{acc}_1)_i + \varphi(\mathrm{acc}_-)_i \\ &= \mathfrak{p}_i + \mathfrak{e}^0_i + \mathfrak{q}_i + \mathfrak{e}^1_i + (-1)^B \left(\mathfrak{p}_{i+\bar{e}} + \mathfrak{e}^0_{i+\bar{e}} - \mathfrak{q}_{i+\bar{e}} - \mathfrak{e}^1_{i+\bar{e}}\right) + \mathfrak{e}_i \\ &= \bar{\mathfrak{p}}_{\bar{i}} + \mathfrak{e}^0_i + \bar{\mathfrak{q}}_{\bar{i}} + \mathfrak{e}^1_i + (-1)^B \left(\bar{\mathfrak{p}}_{\bar{i}} + \mathfrak{e}^0_{i+\bar{e}} - \bar{\mathfrak{q}}_{\bar{i}} - \mathfrak{e}^1_{i+\bar{e}}\right) + \mathfrak{e}_i \\ &= \begin{cases} 2\bar{\mathfrak{p}}_{\bar{i}} + (\mathfrak{e}^0_i + \mathfrak{e}^0_{i+e} + \mathfrak{e}^1_i - \mathfrak{e}^1_{i+e}) + \mathfrak{e}_i & \text{if } B = 0 \\ 2\bar{\mathfrak{q}}_{\bar{i}} + (\mathfrak{e}^0_i - \mathfrak{e}^0_{i+e} + \mathfrak{e}^1_i + \mathfrak{e}^1_{i+e}) + \mathfrak{e}_i & \text{if } B = 1 \end{cases}\end{aligned}$$

where $\bar{i} = \left\lfloor \frac{ki}{N} \right\rceil$ and $\mathfrak{e}$ is an error polynomial induced from blind-rotation and RLWE key-switching. Then,

$$\mathcal{B}_{\mathsf{out}} \leq 2(\mathcal{B}_0 + \mathcal{B}_1) + \mathcal{B}_{BR} + k \cdot \mathcal{B}_{KS}$$

and the result follows. □

Observe that the conditions to use $\mathsf{SIG-MUX}$ and $\mathsf{TWO-MUX}$ are identical, but recall that for $\mathsf{SIG-MUX}$ we needed to assume that the degree of the polynomials need to be less than $\frac{N}{2}$. Hence, by using $\mathsf{SIG-MUX}$ we can choose only between two sets of $\frac{2k}{2} = k$ values, whereas for $\mathsf{TWO-MUX}$ we may select between sets of $2k$ values though at the cost of having a higher output noise. Finally, we also note that the output error behaves differently, which we should consider when choosing between the two. In the next section, we shall see how to utilize MUX gates for our LUT evaluation.

3.3 LUT Evaluation

We are now ready to present our LUT evaluation technique. We give the full specification in Algorithm 8. The algorithm leverages the second muxing algorithm; it is possible to exchange the call to $\mathsf{TWO-MUX}$ by a call to $\mathsf{SIG-MUX}$, provided we only pack half the coefficients into the initial acc^l.

At a high level, the algorithm proceeds as follows: Initially, a subset of bits of size κ is selected and composed into an integer. Then, as illustrated in Sect. 3.1, we evaluate a binary function that will generate all possible outputs conforming to the subset of bits. In the final stage, we use our RLWE MUX gates to iteratively select the correct output bits. Note that in the application of $\mathsf{EvalBinFunc}$ we use a scaling parameter different than 2, introducing a random sign flip. However, recall that while applying $\mathsf{TWO-MUX}$, the output variable corresponds to the double of one of the input variables. Hence, at the end of the final iteration of the selection step, we shall obtain a scaling parameter of $\pm 2^{u-\kappa-1} \cdot \left\lfloor \frac{Q}{2^{u-\kappa}} \right\rceil \approx \left\lfloor \frac{Q}{2} \right\rceil \pmod{Q}$, no matter what the original sign flip was.

Lemma 5. *Let* $(\mathrm{ct}^i_q)_{i\in\langle u\rangle} = \mathrm{LWE}\left(\frac{q}{2}B_i\right)$ *and let* $f : \mathbb{Z}_{2^u} \mapsto \mathbb{Z}_{2^u}$ *where* $\log_2(u) \in \mathbb{N}$ *be a map. Furthermore, let* $\mathcal{B}_{BR}, \mathcal{B}_{KS}$ *be bounds on the variance of blind-rotation and LWE-RLWE key-switching respectively. Then, Algorithm 8 outputs LWE samples* $(\mathrm{ct}^i_{\mathsf{out}})_{i\in\langle u\rangle} = \mathrm{LWE}(\frac{Q}{2}[f(\sum_{j=0}^{u-1} B_j \cdot 2^j)]_i)$ *with variance* $\mathcal{B}_{\mathsf{out}}$ *such that*

$$\mathcal{B}_{\mathsf{out}} \leq 2^{2u-\kappa} \cdot \mathcal{B}_{BR} + 2^u \cdot \mathcal{B}_{KS}\,.$$

Proof. We start by noting that correctness follows from Lemma 1 and Lemma 4. Then, to show the bound on the output variance, it suffices to track the noise of acc^0 in every iteration of the final loop. Furthermore, it is possible to establish a recurrence relation using Lemma 4 for the output variance. However, the bound obtained in this way overestimates the actual variance, since we multiply by $\mathfrak{h}$ during every RLWE key-switch. Therefore, the error induced by blind-rotation will be equal in chunks of the output error, and, similarly to

Algorithm 8: EvalLUT

```
Input: $(\mathrm{ct}_q^i)_{i\in\langle u\rangle} = \mathrm{LWE}\left(\frac{q}{2}B_i\right)$
Input: A map $f : \mathbb{Z}_{2^u} \mapsto \mathbb{Z}_{2^u}$ where $\log_2(u) \in \mathbb{N}$
Input: A parameter $\kappa$
Output: LWE Samples $(\mathrm{ct}_{\mathsf{out}}^i)_{i\in\langle u\rangle} = \mathrm{LWE}(\frac{Q}{2}\beta_i)$ with $\beta_i = [f(\sum_{j=0}^{u-1} B_j \cdot 2^j)]_i$
for $i = 0$ to $2^\kappa - 1$ do
    Define $f^i : \mathbb{Z}_{2^{u-\kappa}} \mapsto \mathbb{Z}_{2^u}$ such that $f^i(x) = f(x \cdot 2^{u-\kappa} + i)$;
end
Define $F(x) = \big([f^0(x)]_0, ..., [f^0(x)]_{u-1}, [f^1(x)]_0, ..., [f^{2^\kappa-1}(x)]_{u-1}\big)$;
/* Composition */
$\mathrm{ct}_{N,Q}^\kappa \leftarrow \mathsf{AssembleInteger}((\mathrm{ct}_q^i)_{i\in\langle\kappa\rangle}, 2^\kappa)$;
$\mathrm{ct}_{N,q}^\kappa \leftarrow \mathsf{ModSwitch}_{Q\to q}(\mathrm{ct}_{N,Q}^\kappa)$;
$\mathrm{ct}_{n,q}^\kappa \leftarrow \mathsf{KeySwitch}_{\vec{S}\to\vec{s}}(\mathrm{ct}_{N,q}^\kappa)$;
/* Output Generation */
$(\mathrm{ct}^0, ..., \mathrm{ct}^{u\cdot 2^{u-\kappa}-1}) \leftarrow \mathsf{EvalBinFunc}(\mathrm{ct}_{n,q}^\kappa, F, C = 2^{u-\kappa-1})$;
$v \leftarrow \left\lceil \frac{u\cdot 2^{u-\kappa}}{2^\kappa} \right\rceil$;
for $l = 0$ to $v - 1$ do
    $\mathrm{acc}^l \leftarrow [\mathfrak{o}, \mathfrak{o}]^\top$;
    for $j = 0$ to $2^\kappa - 1$ do
        $\mathrm{acc}_{\mathsf{tmp}} \leftarrow \mathsf{RLWEKeySwitch}_{\vec{S}\to\mathfrak{s}}(\mathrm{ct}^{l\cdot 2^\kappa + j})$;
        $\mathrm{acc}^l \leftarrow \mathrm{acc}^l + X^{\frac{N}{2^\kappa}j} \cdot \mathfrak{h} \cdot \mathrm{acc}_{\mathsf{tmp}}$;
    end
end
/* Selection */
for $i = \kappa$ to $u - 1$ do
    $w \leftarrow \lceil \frac{v}{2} \rceil$;
    for $l = 0$ to $w - 1$ do
        $\mathrm{acc}^l \leftarrow \mathsf{TWO-MUX}(\mathrm{ct}_q^i, \mathrm{acc}^l, \mathrm{acc}^{l+w})$;
    end
    $v \leftarrow \frac{v}{2}$
end
for $i = 0$ to $u - 1$ do
    $\mathrm{ct}_{\mathsf{out}}^i \leftarrow \mathsf{SampleExtract}(\mathrm{acc}^0, i \cdot \frac{N}{2^\kappa})$;
end
return $(\mathrm{ct}_{\mathsf{out}}^i)_{i\in\langle u\rangle}$;
```

the message components, these terms would partially cancel out in the proof of Lemma 4. Hence, we establish the result by tracking the variance contribution of RLWE key-switching and the one induced by blind-rotation regarding the iteration i, as no other operations take place in the final selection part. We obtain relations

$$\begin{aligned} \mathcal{B}_{KS}^{(i+1)} &\le 2 \cdot \mathcal{B}_{KS}^{(i)} + 2^\kappa \mathcal{B}_{KS}\,; & \mathcal{B}_{KS}^{(0)} &= 0\,; \\ \mathcal{B}_{BR}^{(i+1)} &\le 2 \cdot \mathcal{B}_{BR}^{(i)} + \mathcal{B}_{BR}\,; & \mathcal{B}_{BR}^{(0)} &= 2^{u-\kappa-1}\mathcal{B}_{BR}\,. \end{aligned}$$

The value of $\mathcal{B}_{BR}^{(0)}$ follows from Lemma 1, whereas the initial $\mathcal{B}_{KS}^{(0)}$ is set to 0 as the RLWE key-switch step may be skipped in the **last** iteration, as we output LWE samples and therefore the last switch is not necessary. Then, we have that

$$\begin{aligned}\mathcal{B}_{\mathsf{out}} &\leq \mathcal{B}_{BR}^{(u-\kappa)} + \mathcal{B}_{KS}^{(u-\kappa)} \\ &\leq 2^{\kappa} \cdot (2^{2u-2\kappa} - 1) \cdot \mathcal{B}_{BR} + 2^{\kappa} \cdot (2^{u-\kappa} - 1)\mathcal{B}_{KS} \\ &\leq 2^{2u-\kappa} \cdot \mathcal{B}_{BR} + 2^{u} \cdot \mathcal{B}_{KS} \,.\end{aligned}$$

□

We note that in order to guarantee correctness, the blind-rotation keys and the κ parameter need to be chosen such that the bounds in Sect. 3.2 hold, i.e. $\mathrm{acc}_0, \mathrm{acc}_1$ need to be 2^{κ}-redundant and for Eqn. 8 to hold w.r.t. ct^{i}. Although restrictive, this leads us to the strength of our approach: reconfigurability. As hinted towards in Sect. 3.1 the requirements for each part of the algorithm are quite different w.r.t. to input and output space. Hence, blind-rotation and other parameters can be chosen specifically for each part which can lead to large performance gains. Furthermore, in the final iterations of the selection stage, the redundancy requirement may become weaker, so different parameters may even be chosen for each iteration. This can occur in cases where, e.g., $u << 2^{\kappa}$. Although the concrete number will vary on the setting, we can see that if our parameter set(s) support a plaintext space of size 2^{κ} and $u + \log_2(u) \leq 2\kappa + 1$ the evaluation will need u blind-rotations in the best case and $2u - \kappa$ blind-rotations in the worst case where the selection bits need to be bootstrapped before applying $\mathsf{TWO-MUX}$.

To conclude this section, we give two optimizations of the LUT evaluation algorithm:

- As described in Algorithm 8, we keyswitch all generated LWE samples and use the packed RLWE ciphertexts in $\mathsf{TWO-MUX}$. Looking again at Algorithm 7 we observe that we compute the difference of two accumulators. However, in practice, this means that it is more beneficial to compute the difference of LWE samples before packing them into a RLWE sample in order to decrease the number of necessary LWE-RLWE key-switches.
- In each iteration of the selection step we extract a number of LWE samples. This extraction is necessary to “reset” the offset induced by the LWE noise for the $\mathsf{TWO-MUX}$ application and facilitate the construction of $\mathrm{acc}_0, \mathrm{acc}_1$. However, if the noise level of input bits is low enough, we can skip this extraction and subsequent $\mathsf{RLWEKeySwitch}$. After a MUX step, the coefficients in each half of the polynomial in the RLWE output induce the two accumulators. Hence, we may set $\mathrm{acc}_0 = \mathrm{acc}_{\mathsf{out}}$ and $\mathrm{acc}_1 = -X^{\frac{N}{2}} \mathrm{acc}_{\mathsf{out}}$ and, provided the noise in input bits is sufficiently low, the next MUX application will yield the correct result.

4 AES Transciphering

In this section, we shall discuss how we implement the AES-128 round function homomorphically. In the clear, the AES state is represented by a 4 by 4 matrix with entries in $GF(2^8)$, which can also be viewed as bytes. In our setting, we will encode the state $\mathcal{S}$ as a rank 3 tensor of dimension $4 \times 4 \times 8$ containing LWE samples encrypting the individual bits of the state byte, which are encoded as

$$\mathcal{S}_{i,j,k} = \mathrm{LWE}\left(\Delta_{q,2} \cdot [b_{i,j}]_k\right),$$

where $b_{i,j}$ is the state byte at entry i, j. This encoding makes XOR operations effectively free as they can be computed through the linear homomorphism of LWE samples.

The AES round function consists of four operations. Note that we will not differentiate between the encryption or decryption circuit as the counterparts of the individual steps operate in an identical manner. The four steps are given by

1. AddRoundKey, in which the key schedule is XOR-ed with the state matrix.
2. SubBytes, which evaluates a bijection on the entries of the state matrix.
3. ShiftRows, which is a permutation of the entries of the state matrix.
4. MixColumns, which is given by a linear function over $GF(2^8)$ on the columns of the state matrix.

In Table 1, we give an overview of how the individual steps are implemented in a homomorphic setting.

Table 1: Homomorphic equivalent of the AES round function steps.

Step	FHE Implementation
AddRoundKey	LWE Sample addition
SubBytes	Algorithm 8
ShiftRows	Permutation of the state tensor
MixColumns	LWE Sample addition and Algorithm 9

Regarding the AddRoundKey step, we note that the key schedule can be obtained in two different ways. The first option is that the party holding the AES secret key can provide a set of LWE samples encrypting each bit of the key, and then we can derive the entire schedule by leveraging Algorithm 8. This solution may be rather time intensive. The second solution requires the party holding the symmetric key to derive the key schedule themselves. Then, the bits of the key schedule may be stored in a small set of RLWE samples from which we can extract LWE samples encoding the bits using SampleExtract i.e., Algorithm 1. The memory overhead is rather trivial compared to the otherwise required keys, such as the bootstrapping or key-switching keys. For AES-128 with 10 rounds, this requires storing $\lceil \frac{128\cdot 10}{N} \rceil$ RLWE samples and, e.g., when $N = 2^{10}, \log_2(Q) = 32$ one needs 2 RLWE samples, each containing $2N$ $\log_2(Q)$-bit integers, yielding an overhead of 16KiB[1]. We finally note that related works [TCBS23, WWL+23] similarly assume that the key-schedule is computed in advance.

We use the LUT evaluation algorithm described in the previous section to compute the SubBytes step. In the case of the MixColumns step, we need to evaluate a matrix-vector multiplication over $GF(2^8)$ using bits. Under normal circumstances, the multiplication may require additional bootstrapping operations, but in our setting, one of the operands is given in plaintext. Hence, we can implement this step only using LWE Sample additions (XOR operations) and do not need any AND gates. The whole procedure for the homomorphic Galois multiplication is given in Algorithm 9 and outputs the bits of the result.

We note that the algorithm is not optimal regarding the number of additions. In fact, in many cases, ciphertexts will be added to themselves. If we were operating on bits, this would be no issue, as the result would be zero. However, in our case XOR is implemented via the addition of LWE samples, and adding a ciphertext to itself will yield an encoding of zero but with larger noise. Therefore, any implementation using Algorithm 9 should consider whether a lower noise level is critical to the correctness of the procedure and if so use a circuit with minimal number of XOR operations.

We can observe that the evaluation of the MixColumns circuit requires up to 9 additions for the encryption circuit and 18 additions for the decryption circuit. This difference stems from the scalars used in MixColumns depending on the direction. Hence, performing those additions *before* the final key- and modulus switching step is generally very beneficial. Then, as we operate under a larger modulus, the final modulus switch will nonlinearly decrease the noise, and we will have a higher success rate in decrypting the bits. Finally,

[1]For reference, the LWE to RLWE key-switching key may require around 1GB.

Algorithm 9: GaloisMul

```
Input: $(\mathrm{ct}^i)_{i\in\langle 8\rangle} = \mathrm{LWE}\,(\Delta_{q,2}B_i)$
Input: $\mathfrak{w} \in GF(2)[X]/(X^8 + X^4 + X^3 + X + 1) = GF(2^8)$
Output: $(\mathrm{ct}^i_{\mathsf{out}})_{i\in\langle 8\rangle} = \mathrm{LWE}\,(\Delta_{q,2}B'_i)$
$(\mathrm{ct}^i_{\mathsf{out}}) \leftarrow [\vec{0}, 0]^\top$;
for $i = 0$ to $7$ do
    if $\mathfrak{w}_i = 1$ then
        $\mathrm{ct}^i_{\mathbf{tmp}}, ..., \mathrm{ct}^7_{\mathbf{tmp}} \leftarrow \mathrm{ct}^0, ..., \mathrm{ct}^{7-i}$;
        $\mathrm{ct}^0_{\mathbf{tmp}}, ..., \mathrm{ct}^{i-1}_{\mathbf{tmp}} \leftarrow [\vec{0}, 0]^\top$;
        for $j = 7$ to $7 - i + 1$ do
            $\mathrm{ct}^{j+i \pmod 8}_{\mathbf{tmp}} \leftarrow \mathrm{ct}^{j+i \pmod 8}_{\mathbf{tmp}} + \mathrm{ct}^j$;
            $\mathrm{ct}^{j+i+1 \pmod 8}_{\mathbf{tmp}} \leftarrow \mathrm{ct}^{j+i+1 \pmod 8}_{\mathbf{tmp}} + \mathrm{ct}^j$;
            $\mathrm{ct}^{j+i+3 \pmod 8}_{\mathbf{tmp}} \leftarrow \mathrm{ct}^{j+i+3 \pmod 8}_{\mathbf{tmp}} + \mathrm{ct}^j$;
            $\mathrm{ct}^{j+i+4 \pmod 8}_{\mathbf{tmp}} \leftarrow \mathrm{ct}^{j+i+4 \pmod 8}_{\mathbf{tmp}} + \mathrm{ct}^j$;
        end
        for $j = 0$ to $7$ do
            $(\mathrm{ct}^i_{\mathsf{out}}) \leftarrow (\mathrm{ct}^i_{\mathsf{out}}) + (\mathrm{ct}^i_{\mathbf{tmp}})$;
        end
    end
end
return $\mathrm{ct}_{\mathsf{out}}$
```

it is worth pointing out that each output bit necessitates a different amount of additions, and we can use this fact for the LUT evaluation algorithm (Algorithm 8) by selecting the most noisy bits for the composition step and bits exhibiting a lower noise level for the selection stage.

5 Evaluation

In this section we provide theoretical and practical evaluation of our LUT algorithm applied and compare its performance in the context of transciphering to other recent works.

5.1 Theoretical Evaluation

We first give a theoretical comparison to [GBA21].

In Table 2, we compare the number of blind-rotations and LWE-to-RLWE keyswitches to the tree based method from [GBA21]. Specifically, we target a setting in which we work on binary digits and in which the LUT have as many inputs as output digits, as our algorithm was designed for this case. At a high level, the tree-based algorithm and ours operate similarly. However, there are three key differences.

First, we do not start from a complete set of RLWE samples from which we select the correct output. Instead, our method starts by consuming κ ciphertexts containing bits in order to generate our initial set. Then, using parameter sets that support plaintext spaces larger than $\mathbb{B}$, we can encode significantly more values in our accumulators. This increase leads to a reduction in the number of necessary blind-rotations. Similar results could be obtained by combining [GBA21] with ideas from [CLOT21]. Finally, the tree based method from [GBA21], by default, requires knowledge of the most significant bit of the message in order to circumvent the issue of negacyclity, an issue we do not face.

Table 2: Number of blind-rotations and LWE to RLWE key switches of related works. We annotate the settings in which circuit bootstrapping [CGGI20] is used with "CBS."

Work	#Blind-Rotations	#LWE-RLWE Switches
[GBA21]	$d \cdot (1 + \sum_{i=0}^{d-2} 2^d)$	$d \cdot (2^d - 1)$
[GBA21] (With CBS)	$O(d)$	
Ours ($\mathsf{SIG-MUX}$)	$d - \kappa + \sum_{i=0}^{d-\kappa-1} \left\lceil \frac{d \cdot 2^{d-\kappa-1-i}}{2^\kappa} \right\rceil$	$d \cdot (2^{d-\kappa+1} - 1)$
Ours ($\mathsf{SIG-MUX}$, CBS)	$O(d)$	
Ours ($\mathsf{TWO-MUX}$)	$d - \kappa + \sum_{i=0}^{d-\kappa-1} \left\lceil \frac{d \cdot 2^{d-\kappa-1-i}}{2^{\kappa+1}} \right\rceil$	$d \cdot (2^{d-\kappa} - 1)$
Ours ($\mathsf{TWO-MUX}$, CBS)	$O(d)$	

Table 3: Output variance of related works compared to ours. u denotes the number of digits, $\mathcal{B}_{KS}, \mathcal{B}_{LWE-RLWE}, \mathcal{B}_{BR}$ respectively denote bounds on the variance induced by LWE-LWE key-switching, LWE-RLWE key-switching and blind-rotation respectively.

Work	Output variance (Reported)	Output variance (Actual)
[GBA21]	$(u-1)\mathcal{B}_{KS} + u\mathcal{B}_{BR}$	$u \log(Q)\mathcal{B}_{KS} + u\mathcal{B}_{BR} + 2^u \cdot \mathcal{B}_{LWE-RLWE}$
Ours	$2^{2u-\kappa} \cdot \mathcal{B}_{BR} + 2^u \cdot \mathcal{B}_{LWE-RLWE}$	

Next, we give an overview of the output variance of the tree based method in Table 3. We note that the formula given in Table 6. of [GBA21] w.r.t the output variance of the tree-based method is not correct. More specifically, the authors of [GBA21] only consider the noise induced by key-switching and blind-rotation. Therefore, they do not consider the noise induced by additional LWE-RLWE key-switching caused by the circuit bootstrapping operation and the noise resulting from the public key switch operation in Algorithm 6 in [GBA21].

To end this subsection, we again note that the novel algorithm from [BPR23] may be used to evaluate binary LUTs more efficiently. However, a proper comparison is limited because the authors do not provide bounds on the worst-case umber of blind-rotations and other operations or on the output variance.

5.2 Experiments

In this section, we evaluate our technique and implement the AES encryption circuit utilizing Algorithm 8 and the natural homomorphism of LWE samples.

We run our experiments on an Intel i7 13800H 5.2 GHz processor with 32 GB of RAM and implement our LUT evaluation technique using the OPENFHE library[2] modified that it exposes the ring secret key. We compile the library using Clang-15 and always use the `WITH_NATIVEOPT=ON` flag and the `BUILD_STATIC=ON` in order to leverage link time optimization. Depending on the size of the ring moduli, we also supply the `WITH_NATIVESIZE=64` or the `WITH_NATIVESIZE=32` flags to decrease the memory usage. OPENFHE does not provide an LWE to RLWE keyswitch operation, hence we implement our own optimized implementation that takes the primary bottlenecks into account, namely cache misses and unnecessary data loads and write-backs.

We give our parameter sets used in Table 4. For each parameter set, we give the the security[3] and failure probability in Table 5. Furthermore, we note that we give the failure probability with regards the blind-rotation algorithm from [LMK+23], always using 10 automorphism keys, and that we leverage sparse LWE keys to control the noise of modulus switching, as has been done in other works [KS22]. However, in the implementation, we

[2]https://github.com/openfheorg/openfhe-development
[3]Established using the lattice estimator https://github.com/malb/lattice-estimator

Table 4: Parameter sets employed. σ is the standard deviation employed for R(LWE) samples, $||\vec{s}||_0$ is the Hamming weight of $\vec{s}$ and n_{BR} is the number of bootstraps per AES round function evaluation. L_{boot} is the decomposition basis for the algorithm in [LMK+23]; note that we give three values for L_{boot}, one for each stage of Algorithm 8.

Set	$\log(Q)$	q	n	N	σ	L_{ksk}	$L_{\mathsf{ksk,rlwe}}$	L_{boot}	$\lVert\vec{s}\rVert_0$	n_{BR}
I	28	2^{10}	512	2^{10}	3.19	2^5	2^6	$2^7, 2^3, 2^4$	100	128
II	28	2^{10}	600	2^{10}	3.19	2^5	2^6	$2^7, 2^4, 2^{10}$	100	176
III	36	2^{10}	448	2^{10}	3.19	2^5	2^6	$2^{12}, 2^9, 2^{12}$	100	128
IV	42	2^{11}	600	2^{11}	3.19	2^7	2^7	2^{14}	100	128

Table 5: Target failure probabilities. We also give the number of evaluations before the failure probability exceeds 50% and the corresponding amount of data transciphered.

	I	II	III	IV
Security	128	128	80	128
Failure probability $\mathbb{P}[\mathsf{FAIL\text{-}ROUND}]$	2^{-8}	2^{-15}	2^{-22}	2^{-63}
Failure probability $\mathbb{P}[\mathsf{FAIL\text{-}BLOCK}]$	2^{-5}	2^{-12}	2^{-19}	2^{-61}
n_{eval} until $\mathbb{P}[\mathsf{FAIL\text{-}ROUND}] > 0.5$	177	$2.2 \cdot 10^5$	$2.9 \cdot 10^6$	$8 \cdot 10^{17}$
Equiv. amount of data in Bytes	$2.76 \cdot 10^3$	$3.54 \cdot 10^5$	$4.5 \cdot 10^6$	$1.28 \cdot 10^{19}$

use the blind-rotation technique by Chilotti et al. [CGGI20] modified for ternary secrets in [MP21], which will have similar timings but significantly larger noise growth. This is due to the fact that OPENFHE does not support encrypted accumulators for the blind-rotation technique from [LMK+23] and does not properly implement the round-to-odd technique by Liu et al. [LMK+23] or Algorithm 6 from [LMK+23][4]. As the implementation is not exactly the same, there is a larger error which can be neglected in some cases. However, we chose our parameters with regard to the variance of the blind-rotation of [LMK+23] and therefore the error may cause the LUT evaluation to fail. We briefly discuss how our parameter sets were obtained. We always fix the standard deviation $\sigma = 3.19$. Initially we determine tuples (Q_i, N_i) with $N_i \in \{2^{10}, 2^{11}\}$ for the RLWE instantiation based on security considerations. Then, we proceed by determining a set of acceptable bases L for the blind-rotation keys for each stage, requiring $\log_L(Q) \leq 10$ when $\log_2(Q) < 32$ and $\log_L(Q) \leq 4$ when $\log_2(Q) < 64$ to obtain a set of parameters that are efficient. Then, we test every possible combination of L bases and compute the output variance using Lemma 5. Finally, we select the combination of bases that induces a desired failure probability while also minimizing the number of digits during blind-rotation, that is $\log_L(Q)$.

The security and failure probability were chosen so that Set I & III were comparable to the parameters used in the related works. Parameter Set II was chosen to provide a compromise: recall from Section 3 that we can choose to bootstrap the bits used in the final selection stage of our LUT algorithm, decreasing the noise of the encoded. Therefore, the probability of success of the MUX algorithms increases, but at the cost of requiring more time. Finally, parameter Set IV was selected to explore the performance of our LUT evaluation in light of recent attacks on FHEW based schemes that exploit a higher failure probability [CSBB24, CCP+24].

The security and failure probability of related works are given in Table 6. We note that these may differ from the values report in the individual works which can be attributed to updates to the Lattice estimator. For [WWL+23] no failure probability was reported and we give our own estimate based on the formula of the output variance given in [WWL+23]. For [TCBS23], we briefly note that there is a significant difference between the security level reported by the authors and the value we obtained using the lattice estimator. In

[4]See https://github.com/openfheorg/openfhe-development/pull/338

Table 6: Security and failure probability of related works. In brackets: values reported in the corresponding work.

	[TCBS23]	[WWL+23]	[BPR23]
Security in bits	27 (128)	110 (128)	104 (128)
$\mathbb{P}[\mathsf{FAIL}]$	2^{-23}	2^{-7}	2^{-40}

Table 7: LUT evaluation time in seconds for 8 bits and $\kappa = 5$.

Set	I	II	III	IV
LUT evaluation time	0.383	0.470	0.414	0.753
AES transciphering time	62	76	67	121

[TCBS23], the authors use the original TFHE library[5] which discretizes the unit torus $\mathbb{R}/\mathbb{Z}$ onto a 32 bit integer and uses binary (R)LWE keys. Hence, the modulus is in practice equal to 2^{32} and the authors use a Gaussian width parameter of $\alpha = 9.6 \cdot 10^{-11}$ yielding in practice a standard deviation $\sigma = \alpha \cdot \frac{q}{\sqrt{2\pi}} \approx 0.16$ (cf. [ACF+15]). Together with an LWE dimension of 2^{11}, these parameters are susceptible to an attack using Gröbner bases from [ACF+15]. In our experiments, we use $\kappa = 5$ as, on the one hand, it allows us to evaluate an 8-bit LUT in 8 blind-rotations, and on the other hand, it is not too difficult to find appropriate parameter sets. Furthermore, in all cases we will rely on the $\mathsf{TWO} - \mathsf{MUX}$ algorithm. If we relied on $\mathsf{SIG} - \mathsf{MUX}$ instead, the first iteration of the selection stage would need double the number of blind-rotations as we can pack a smaller amount of values into the accumulator. Such arguments will generally decide which technique to use: If finding appropriate parameters at a certain failure rate is difficult, it may be worth sacrificing efficiency for a lower growth of the error variance using $\mathsf{SIG} - \mathsf{MUX}$. On the other hand, if we can be flexible with parameters but wish to focus on an efficient implementation, then we recommend leveraging $\mathsf{TWO} - \mathsf{MUX}$ as the higher noise growth can be mitigated by using larger moduli.

In Table 7 we give the time necessary per LUT evaluation for an 8 to 8 bit look-up table and the time necessary for an evaluation of the AES-128 decryption circuit for all our parameters. We observe that Set II is faster than Set I despite a larger number of bootstraps due to larger decomposition bases. Set IV yields the slowest run-time as it necessitated a dimension $N = 2^{11}$.

In Table 8, we compare our timings. We note that the related works leverage the Torus variant of the CGGI/GINX bootstrapping. This implies that the implementations leverage a power-of-two ciphertext and ring modulus and consequently, no modular reductions need to be performed as x86 processors are able to naturally operate modulo 2^{32} or 2^{64}, saving a considerable amount of time. We note that our algorithms may also be instantiated in this way and would yield another speedup. In the case of [WWL+23], usage of the TFHE-PP library[6] implies that many parameters are provided at compile time, which yields some additional performance gains. We give both the reported timings and timings that were adjusted based on the ratio between the clock speed of the processor employed in the related works and our processor. No further comparisons were possible as none of the related works chose to publish their implementation.

Considering the previous points, we finally note that despite the aforementioned advantages of related works, our methodology still yields the fastest execution time for the decryption circuit, outperforming [TCBS23] by a factor of 4 in the best case. Likewise, we show faster execution time than [WWL+23] by about 19% with a similar failure rate (Set I) and 12% in the best case (Set III).

[5]`tfhe.github.io`
[6]`https://github.com/virtualsecureplatform/TFHEpp.git`

Table 8: Comparison of recent transciphering works. In case of related works we give the reported timings, followed by the timings that would result from an execution on our testbed.

Work	Core Technique	Torus Instantiation	Sequential Timing (s.)
[TCBS23]	LUT Evaluation from [GBA21]	Yes	270 (244)
[WWL$^+$23]	Circuit Bootstrapping using [CLOT21, CIM19] and Packing from [CGGI20]	Yes	87 (76)
[BPR23]	Novel framework to evaluate boolean functions	Yes	109 (89)
Ours	Algorithm 8	No	62 (Set I) 76 (Set II) 67 (Set III) 121 (Set IV)

Finally, we observe that the fourth parameter set yields a slower execution than [WWL$^+$23, BPR23], which is to be expected as this parameter set has the lowest failure rate out of all related work by a significant margin, requiring larger moduli and ring dimensions.

5.3 Comparison to FHE-Friendly Ciphers

To close the evaluation section, we compare our results on the transciphering of AES-128 to recent works implementing transcipherings of schemes that are considered to be FHE friendly, namely, Trivium [DC06], Kreyvium [CCF$^+$18], Filip [MCJS19] and Elisabeth [CHMS22]. The aforementioned schemes are stream ciphers. We note that FHE-friendly block ciphers exist, such as Chaghri [AMT22], but to the best of our knowledge, there are no works evaluating the transciphering of such schemes using accumulator-based bootstrapping algorithms.

An overview of the related works is given in Table 9 and we note that they were chosen due to fact that, similarly to our methodology, they rely on accumulator-based FHE schemes. We note that no sequential timings were given in the case of [BOS23]. Furthermore, [BOS23] relies on an implementation of TFHE whereas [MPP24] leverages the novel FINAL blind-rotation [BIP$^+$22].

We give the throughput per bit of our AES-128 transciphering in Table 10. Although we did not parallelise our implementation, we note that the 16 S-BOX evaluations in each round are completely independent and could therefore be perfectly parallelised. Hence, we also give the (hypothetical) throughput for this case. By contrasting Table 9 and Table 10 we observe that, despite the speedups gained through our methodology in the context of AES transciphering, the throughput is still lacking compared to the transciphering of FHE-friendly ciphers. By default, the transciphering of AES remains 2-3 orders of magnitudes slower, and even in a setting where we would achieve an ideal speedup through parallelisation, our performance remains an order of magnitude slower on average.

The drastic differences in throughput indeed justify a faster adoption of FHE-friendly ciphers. However, we also note that one may not be able to leverage them in every scenario. Such an example is given in a setting where the AES encrypted data is already stored remotely. Then, the party holding the symmetric key would first need to transform the AES encryption into the encryption w.r.t. a FHE-friendly cipher. However, it will be more

Table 9: Throughput of FHE friendly ciphers in milliseconds per bit. We also state the timings adjusted to the processor clock frequency of our testbed. We note that the measurements for Elisabeth in [MPP24] stem from [CHMS22].

Work	[BOS23]		[MPP24]	
Parallelization	Yes		No	
Failure probability	2^{-40}		2^{-30}	
Scheme	Trivium	Kreyvium	Filip	Elisabeth
Throughput	1.8	2.3	6.3	22.7
Throughput (adjusted)	1.2	1.5	3.6	3.6

Table 10: Throughput of the AES-128 transciphering in our work in milliseconds per bit. We also state the timing we would obtain by performing the 16 S-BOX evaluations in parallel every round.

Parameter Set	I	II	III	IV
Throughput	484	593	523	1002
Throughput (parallel)	30	37	32	62

convenient to perform the transciphering directly in this case.

6 Conclusion and Future Work

In this work we described how to evaluate look-up tables based on binary digits efficiently by almost exclusively relying on the well-known building blocks of TFHE-like schemes. Despite its simplicity, our method exhibits efficiency rivalling and exceeding concurrent works, and at the same time, our method is highly adaptive. However, certain topics still need to be investigated further, which we leave to future work. More concretely, we may significantly boost performance by leveraging the blind-rotation techniques based on the NTRU assumption as introduced in FINAL by Bonte et al. [BIP+22] or NTRU-ν-ium by Kluczniak [Klu22]. Furthermore, unlike other methods, blind-rotation efficiency was not the only bottleneck of our method, but LWE to RLWE key-switching as well, which could be significantly sped up by exploiting special hardware such as GPUs or FPGAs.

Acknowledgements

This work is supported by the Flemish Government through FWO SBO project MOZAIK S003321N. It was also funded in part by the European Network of Excellence dAIEDGE under Grant Agreement Nr. 101120726 and by CyberSecurity Research Flanders with reference number VR20192203.

References

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic Algorithms for LWE Problems. *ACM Commun. Comput. Algebra*, 49(2):62, aug 2015.

[AMT22] Tomer Ashur, Mohammad Mahzoun, and Dilara Toprakhisar. Chaghri - A FHE-Friendly Block Cipher. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 139–150. ACM, 2022.

[BIP$^+$22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE Instantiated with NTRU and LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022*, pages 188–215. Springerd, 2022.

[BOS23] Thibault Balenbois, Jean-Baptiste Orfila, and Nigel Smart. Trivial transciphering with trivium and tfhe. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '23, page 69–78, New York, NY, USA, 2023. Association for Computing Machinery.

[BPR23] Nicolas Bon, David Pointcheval, and Matthieu Rivain. Optimized Homomorphic Evaluation of Boolean Functions. Cryptology ePrint Archive, Paper 2023/1589, 2023. https://eprint.iacr.org/2023/1589.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, 2011.

[CCF$^+$18] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. *Journal of Cryptology*, 31(3):885–916, Jul 2018.

[CCP$^+$24] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the indcpa-d security of exact fhe schemes. Cryptology ePrint Archive, Paper 2024/127, 2024. https://eprint.iacr.org/2024/127.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, 33(1):34–91, Jan 2020.

[CHMS22] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards case-optimized hybrid homomorphic encryption. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 32–67, Cham, 2022. Springer Nature Switzerland.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 106–126. Springer, 2019.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, pages 670–699. Springer, 2021.

[CSBB24] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical cpad security of "exact" and threshold fhe schemes and libraries. Cryptology ePrint Archive, Paper 2024/116, 2024. https://eprint.iacr.org/2024/116.

[DC06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security*, pages 171–186, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[DEG$^+$18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit. In *CRYPTO 2018, Proceedings, Part I*, page 662–692. Springer, 2018.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, pages 617–640. Springer, 2015.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):229–253, Feb. 2021.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013*, pages 75–92. Springer, 2013.

[Klu22] Kamil Kluczniak. NTRU-v-Um: Secure Fully Homomorphic Encryption from NTRU with Small Modulus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1783–1797. ACM, 2022.

[KS22] Kamil Kluczniak and Leonard Schild. FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):501–537, Nov. 2022.

[LMK$^+$23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW Bootstrapping with Small Evaluation Keys, and Applications to Threshold Homomorphic Encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023*, pages 227–256. Springer, 2023.

[LMP23] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping. In *ASIACRYPT 2022, Proceedings, Part II*, page 130–160. Springer, 2023.

[MCJS19] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators for efficient fhe: Better instances and implementations. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology – INDOCRYPT 2019*, pages 68–91, Cham, 2019. Springer International Publishing.

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like Cryptosystems. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '21, page 17–28. ACM, 2021.

[MPP24] Pierrick Méaux, Jeongeun Park, and Hilder V. L. Pereira. Towards practical transciphering for FHE with setup independent of the plaintext space. *IACR Communications in Cryptology*, 1(1), 2024.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[TCBS23] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. 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*, WAHC '23, page 79–90. ACM, 2023.

[WWL$^+$23] Benqiang Wei, Ruida Wang, Zhihao Li, Qinju Liu, and Xianhui Lu. Fregata: Faster Homomorphic Evaluation of AES via TFHE. In Elias Athanasopoulos and Bart Mennink, editors, *Information Security*, pages 392–412. Springer, 2023.