Published in S. Dolev, J. Katz, and A. Meisels, Eds., *Cyber Security, Cryptography, and Machine Learning (CSCML 2022)*, vol. 13301 of *Lecture Notes in Computer Science*, pp. 1–18, Springer, 2022.

Blind Rotation in Fully Homomorphic Encryption with Extended Keys

Marc Joye and Pascal Paillier

Zama, France

Abstract. Most solutions for fully homomorphic encryption rely on hard lattice problems. Accordingly, the resulting ciphertexts must contain a certain level of noise to guarantee the security of the encryption. Running homomorphic operations on these noisy ciphertexts in turn further increases the noise level in the resulting ciphertexts. If the noise exceeds a given threshold, the ciphertexts are no longer decryptable. Bootstrapping enables to deal with this issue by resetting the noise present in a ciphertext to a nominal level.
Certain fully homomorphic encryption schemes require the use of binary keys for the bootstrapping operation. This paper describes how to extend the underlying blind rotation so as to efficiently support a wider number of key formats. It also investigates a multi-digit approach wherein multiple key digits are processed concurrently. All in all, the proposed solutions offer more flexibility in the parameter selection and yield a variety of new trade-offs for better performance.

Keywords: Fully homomorphic encryption · FHEW · TFHE · Key generation · Private machine learning.

1 Introduction

Fully homomorphic encryption (FHE) [16] is a very powerful cryptographic primitive. It allows performing arbitrary computation directly on encrypted data—without ever requiring intermediate decryption.

All known FHE instantiations follow the same blueprint, as originally devised in Gentry's seminal paper [11]. The idea is (i) to start with a "somewhat" homomorphic encryption scheme which supports a bounded number of homomorphic operations and (ii) to convert it into a *fully* homomorphic encryption scheme. The conversion step is referred to as the *bootstrapping*. Basically, this is accomplished by homomorphically evaluating the decryption function on the ciphertext. The resulting ciphertext encrypts the same plaintext but is somehow "refreshed". Homomorphic operations can therefore be further iterated again and again. This is particularly appealing in the case of private machine learning, where inference is performed homomorphically over user-encrypted data using substantially deep models with a number of layers *in the hundreds*.

The bootstrapping is however a relatively demanding operation and, despite of being intensively studied (e.g., [12,2,6,3,9,10,7,17]), remains the main bottle-

neck in current FHE implementations. Another approach is to avoid bootstrappings altogether and to increase the cryptographic parameters accordingly in order to accommodate the circuit being evaluated in a *leveled* way [5]. This, in turn, limits homomorphic inference to smaller models with at most a few tens of layers.

Our contributions. The most efficient bootstrappings to date are achieved by GSW-derived cryptosystems [9,7] and their variants. The TFHE cryptosystem [7] displays a time of a few tens of milliseconds to perform a bootstrapping with typical parameters on a regular laptop. By design, TFHE (and its variants like [8]) requires binary keys in an essential way for the bootstrapping.

In a recent work, Micciancio and Polyakov [15] remark that a ternary key can be viewed as a difference of two binary keys. This allows them to evaluate a bootstrapping with ternary keys as a series of two original bootstrappings (*i.e.*, with binary keys). Moving from binary keys to ternary keys or more can be useful at it allows for trade-offs (speed/size of the bootstrapping keys/noise level) otherwise not necessarily applicable for a given security level.

Another important motivation to switch from binary to ternary (or more) is to easily fall back to the next best alternative, should *e.g.* the special case of binary keys reveal less secure than expected. Our extended settings allow one to carefully reevaluate the new best trade-offs available after any kind of security-impacting breakthrough.

Our main results are:

- The core operation for the bootstrapping in TFHE and the likes is a blind rotation. It consists of a succession of *external products* which comprise polynomial multiplications and integer recodings. If n denotes the number of digits of a ternary key, the method of Micciancio–Polyakov requires $2n$ external products. We rely on an additive representation and show how a bootstrapping with ternary keys can be done with only n external products. Furthermore, our method features the same memory requirements as in Micciancio–Polyakov's method for storing the corresponding bootstrapping keys.
- We demonstrate that our approach is generic and can be adapted to support arbitrary key formats. Specifically, we extend our ternary approach to keys expressed in a general radix. Somewhat surprisingly, for an encryption key represented with n digits, the number of external products required to complete a bootstrapping is equal to n. We note that the dimension n is a decreasing function of the radix.
- In [17], Zhou *et al.* point out that two bits of the secret key can be processed concurrently during the bootstrapping of the TFHE scheme. Their approach was later refined in [4]. We extend our additive splitting to higher radices and detail how the number of external products can be reduced to n/d by processing d digits per iteration. The number of bootstrapping keys however increases very quickly. In practice, d is likely restricted to small values like $d = 2$ or $d = 3$.

Outline of the paper. The rest of the paper is organized as follows. In the next section, we review the bootstrapping behind TFHE, including its extension to programmable bootstrapping. Section 3 introduces our main technique. We explain how the multiplicative splitting in Micciancio–Polyakov's approach can be turned into an additive one. This gives rise to a more efficient bootstrapping for ternary keys. We then generalize our new approach to higher-radix representations in Section 4. We also cover the case of multiple digits processed concurrently. Section 5 provides a performance analysis according to the parameter selection. Finally, we conclude the paper in Section 6.

2 Programmable Bootstrapping

The bootstrapping is an essential technique for FHE as it enables to control the noise growth and to refresh ciphertexts whenever the noise exceeds a given level. In this section, we review the bootstrapping for the TFHE family and its extension to *programmable* bootstrapping.

The (discretized) TFHE scheme. For our purposes, we consider the TFHE family. We follow the presentation of [8,14].[1] Define the polynomial ring $\mathbb{Z}_{N,q}[X] = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N+1)$ where q and N are powers of 2. Define also the binary set $\mathbb{B} = \{0,1\}$ and $\mathbb{B}_N[X] = \mathbb{B}[X]/(X^N+1)$. For a secret key $\mathfrak{z} \xleftarrow{\$} \mathbb{B}_N[X]^k$, the GLWE encryption of a plaintext $\bar{\mu} \in \mathbb{Z}_{N,q}[X]$ is given by $\overline{\boldsymbol{c}} = (\overline{a}_1, \ldots, \overline{a}_k, \overline{b}) \in \mathbb{Z}_{N,q}[X]^{k+1}$ where $\overline{a}_j \xleftarrow{\$} \mathbb{Z}_{N,q}[X]$ and $\overline{b} = \sum_{j=1}^{k} \mathfrak{z}_j\, \overline{a}_j + \bar{\mu}^*$ with $\bar{\mu}^* = \bar{\mu} + \bar{e}$ for some (small) random noise $\bar{e}$. We write $\overline{\boldsymbol{c}} \leftarrow \mathrm{GLWE}_{\mathfrak{z}}(\bar{\mu})$. When $(N,k) = (1,n)$, it turns out that $\mathbb{Z}_{N,q}[X] = \mathbb{Z}/q\mathbb{Z}$ and the above procedure leads to an LWE ciphertext. We then write $\bar{\boldsymbol{c}} \leftarrow \mathrm{LWE}_{\boldsymbol{s}}(\bar{\mu}) = (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$ as the encryption of a plaintext $\bar{\mu} \in \mathbb{Z}/q\mathbb{Z}$ under the secret key $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathbb{B}^n$, where $\bar{a}_j \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ and $\bar{b} = \sum_{j=1}^{n} s_j\, \bar{a}_j + \bar{\mu}^*$ with $\bar{\mu}^* = \bar{\mu} + \bar{e}$ for some (small) random noise $\bar{e}$.

Blind rotation. The main step in the TFHE bootstrapping is the so-called blind rotation. It converts an LWE ciphertext $\bar{\boldsymbol{c}} \leftarrow \mathrm{LWE}_{\boldsymbol{s}}(\bar{\mu}) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$ into a ciphertext $\overline{\boldsymbol{c}}' \leftarrow \mathrm{GLWE}_{\mathfrak{z}'}(X^{-\tilde{\mu}^*} \cdot \overline{v}) \in \mathbb{Z}_{N,q}[X]^{k+1}$; namely, a GLWE encryption of $X^{-\tilde{\mu}^*} \cdot \overline{v}$ under key $\mathfrak{z}' \in \mathbb{B}_N[X]^k$, where $\tilde{\mu}^*$ is a rounded approximation of $\bar{\mu}^* = \bar{\mu} + \bar{e}$ and $\overline{v}$ is a "test" polynomial. Specifically, if $\bar{\boldsymbol{c}} = (\bar{a}_1, \ldots, \bar{a}_n, \bar{b})$ then

$$-\tilde{\mu}^* = -\tilde{b} + \sum_{j=1}^{n} s_j\, \tilde{a}_j \pmod{2N}$$

where

$$\begin{cases} \tilde{b} = \left\lceil \frac{2N(\bar{b} \bmod q)}{q} \right\rfloor \\ \tilde{a}_j = \left\lceil \frac{2N(\bar{a}_j \bmod q)}{q} \right\rfloor \quad (1 \le j \le n) \end{cases}$$

[1] As originally described, TFHE is defined over the real torus $\mathbb{R}/\mathbb{Z}$. We rather consider the discretized torus $q^{-1}\mathbb{Z}/\mathbb{Z}$ and identify its elements with integers modulo q.

and the test polynomial $\overline{v}$ is programmed as a look-up table so that $X^{-\tilde{\mu}^*} \cdot \overline{v}$, up to a random indexing error (*a.k.a.* drift), encodes $f(\bar{\mu})$ for a chosen function f; see [8,14] for details.

The blind rotation is detailed in Algorithm 1. It requires n bootstrapping keys, $\mathsf{bsk}[j] \leftarrow \mathrm{GGSW}_{\mathfrak{z}'}(s_j)$ for $1 \leq j \leq n$; GGSW denotes a (general) Gentry–Sahai–Waters encryption [13] derived from a gadget matrix $\mathbf{G}$. The blind rotation can be calculated as a series of CMux operations. On input a GGSW ciphertext $\overline{\mathscr{C}}$ encrypting a bit $\mathrm{b} \in \{0,1\}$ and two GLWE ciphertexts $\overline{\boldsymbol{c}}_{\mathbf{0}}$ and $\overline{\boldsymbol{c}}_{\mathbf{1}}$, the CMux operation outputs a ciphertext encrypting the same plaintext as $\overline{\boldsymbol{c}}_{\mathbf{b}}$,

$$\overline{\boldsymbol{c}}' \leftarrow \mathrm{CMux}(\overline{\mathscr{C}}, \overline{\boldsymbol{c}}_{\mathbf{0}}, \overline{\boldsymbol{c}}_{\mathbf{1}}) \coloneqq \overline{\boldsymbol{c}}_{\mathbf{0}} + \overline{\mathscr{C}} \boxdot (\overline{\boldsymbol{c}}_{\mathbf{1}} - \overline{\boldsymbol{c}}_{\mathbf{0}})$$

where $\boxdot$ denotes the external product of ciphertexts. The external product dominates the cost in a blind rotation. Given a GGSW ciphertext $\mathscr{C}_{\mathbf{1}} \leftarrow \mathrm{GGSW}(\mu_1)$ and a GLWE ciphertext $\boldsymbol{c}_{\mathbf{2}} \leftarrow \mathrm{GLWE}(\mu_2)$, their external product is defined as $\mathscr{C}_{\mathbf{1}} \boxdot \boldsymbol{c}_{\mathbf{2}} = G^{-1}(\boldsymbol{c}_{\mathbf{2}}) \cdot \mathscr{C}_{\mathbf{1}}$ where $G^{-1}(\boldsymbol{c}_{\mathbf{2}})$ is the gadget decomposition of $\boldsymbol{c}_{\mathbf{2}}$. The transformation G^{-1} flattens the vector of polynomials $\boldsymbol{c}_{\mathbf{2}} \in \mathbb{Z}_{N,q}[X]^{k+1}$ into a row vector of $(k+1)\ell$ polynomials of $\mathbb{Z}[X]/(X^N+1)$ with small signed coefficients. The goal is to contain the noise growth. For better efficiency, the underlying polynomial multiplications are carried out in the Fourier domain (dyadic multiplications). Most of the time in an external product of ciphertexts is spent in going back and forth in the Fourier domain. Again, we refer the reader to [8,14] for details.

Algorithm 1: Blind rotation (binary case).

```
1 acc ← (0, ..., 0, X^{-b̃} · v̄)
2 for j = 1 to n do
3 |   acc ← CMux(bsk[j], acc, X^{ã_j} · acc)
  |       /* acc ← acc + bsk[j] ⊡ ((X^{ã_j} − 1) · acc)        */
4 end for
5 return acc
```

It can be verified that at the end of the for-loop in Algorithm 1, the accumulator acc contains a GLWE encryption of $X^{-\tilde{\mu}^*} \cdot \overline{v}$ under key $\mathfrak{z}'$.

Proposition 1. *Algorithm 1 is correct.*

Proof. First, at initialization, acc contains $(0, \ldots, 0, X^{-\tilde{b}} \cdot \overline{v}) \in \mathbb{Z}_{N,q}[X]^{k+1}$, which is a valid GLWE encryption of $X^{-\tilde{b}} \cdot \overline{v}$. We so have $\overline{\boldsymbol{c}}'_{\mathbf{0}} \coloneqq (0, \ldots, 0, X^{-\tilde{b}} \cdot \overline{v}) = \mathrm{GLWE}_{\mathfrak{z}'}(X^{-\tilde{b}} \cdot \overline{v})$. Next, by induction, we assume that the result is correct

for $i = j - 1$. We must prove that it remains correct for $i = j$. We have:

$$\begin{aligned}
\overline{\boldsymbol{c}}'_{\boldsymbol{j}} &\coloneqq \mathrm{CMux}(\mathsf{bsk}[j], \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}}, X^{\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}}) \\
&= \begin{cases} \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}} & \text{if } s_j = 0 \\ X^{\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}} & \text{if } s_j = 1 \end{cases} \\
&= X^{s_j\,\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}} \\
&= X^{s_j\,\tilde{a}_j} \cdot \mathrm{GLWE}_{\mathfrak{z}'}(X^{-\tilde{b}+\sum_{i=1}^{j-1} s_i\,\tilde{a}_i} \cdot \overline{v}) = \mathrm{GLWE}_{\mathfrak{z}'}(X^{-\tilde{b}+\sum_{i=1}^{j} s_i\,\tilde{a}_i} \cdot \overline{v})
\end{aligned}$$

which proves the correctness of Algorithm 1. □

3 Using Ternary Keys

As seen in the above proof, the bootstrapping for the TFHE family crucially makes use of the identity

$$\begin{aligned}
X^{s_j\,\tilde{a}_j} &= \begin{cases} 1 & \text{if } s_j = 0 \\ X^{\tilde{a}_j} & \text{if } s_j = 1 \end{cases} \\
&= s_j\,(X^{\tilde{a}_j} - 1) + 1
\end{aligned}$$

for $s_j \in \{0, 1\}$. It is therefore inherently restricted to binary keys. This section exposes two different strategies to extend TFHE and the likes to ternary keys.

3.1 Micciancio–Polyakov's Approach

In [15], Micciancio and Polyakov astutely notice that any vector $\boldsymbol{s} = (s_1, \ldots, s_n)$ with ternary entries $s_j \in \{0, 1, -1\}$, $1 \leq j \leq n$, can be expressed as the difference of two binary vectors $\boldsymbol{s^1} = (s_1^1, \ldots, s_n^1)$ and $\boldsymbol{s^2} = (s_1^2, \ldots, s_n^2) \in \mathbb{B}^n$:

$$(s_1, \ldots, s_n) = (s_1^1, \ldots, s_n^1) - (s_1^2, \ldots, s_n^2)$$

where, for $1 \leq j \leq n$,

$$(s_j^1, s_j^2) = \begin{cases} (0,0) & \text{if } s_j = 0 \\ (1,0) & \text{if } s_j = 1 \\ (0,1) & \text{if } s_j = -1 \end{cases}.$$

The number of bootstrapping keys is doubled and is equal to $2n$. They are given by

$$\mathsf{bsk}[2(j-1)+i] \leftarrow \mathrm{GGSW}_{\mathfrak{z}'}(s_j^i)$$

for all $1 \leq j \leq n$ and for all $1 \leq i \leq 2$. With the previous notation, the authors of [15] exploit the multiplicative nature of the bootstrapping. The GLWE

encryption of $X^{-\tilde{b}+\sum_{j=1}^{n} s_j\,\tilde{a}_j} \cdot \overline{v} = \left(\prod_{j=1}^{n} X^{s_j^1\,\tilde{a}_j}\,(X^{-1})^{s_j^2\,\tilde{a}_j}\right) \cdot (X^{-\tilde{b}} \cdot \overline{v})$ is then obtained iteratively as

$$\begin{cases} \overline{\boldsymbol{c}}'_{\boldsymbol{2j-1}} \leftarrow \mathrm{CMux}(\mathsf{bsk}[2j-1], \overline{\boldsymbol{c}}'_{\boldsymbol{2j-2}}, X^{\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{2j-2}}) \\ \overline{\boldsymbol{c}}'_{\boldsymbol{2j}} \leftarrow \mathrm{CMux}(\mathsf{bsk}[2j], \overline{\boldsymbol{c}}'_{\boldsymbol{2j-1}}, X^{-\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{2j-1}}) \end{cases}$$

for $j = 1, \ldots, n$, with $\overline{\boldsymbol{c}}'_{\boldsymbol{0}} \leftarrow (0, \ldots, 0, X^{-\tilde{b}} \cdot \overline{v})$. Algorithmically, we have:

Algorithm 2: Blind rotation (ternary case).

```
acc ← (0, ..., 0, X^{-b̃} · v̄)
for j = 1 to n do
    acc ← acc + bsk[2j − 1] ⊡ ((X^{ã_j} − 1) · acc)
    acc ← acc + bsk[2j] ⊡ ((X^{−ã_j} − 1) · acc)
end for
return acc
```

With ternary keys, the evaluation of $\mathrm{GLWE}_{\mathfrak{z}'}\left(X^{-\tilde{b}+\sum_{j=1}^{n} s_j\,\tilde{a}_j} \cdot \overline{v}\right)$ thus involves $2n$ external products. For a same value of n, this is twice more than in the binary case.

3.2 Proposed Approach

The blind rotation with ternary keys of Algorithm 2 can be largely improved. Instead of a multiplicative split, we observe that the monomial $X^{s_j\,\tilde{a}_j} = X^{(s_j^1 - s_j^2)\,\tilde{a}_j}$ with $s_j^1, s_j^2 \in \{0, 1\}$ can be expressed in an *additive* way as

$$X^{s_j\,\tilde{a}_j} = s_j^1\,(X^{\tilde{a}_j} - 1) + s_j^2\,(X^{-\tilde{a}_j} - 1) + 1 \ . \tag{1}$$

Remark 1. It is interesting to note that Equation (1) can be equivalently expressed as $X^{s_j\,\tilde{a}_j} = (1 - s_j^1 - s_j^2) + s_j^1\,X^{\tilde{a}_j} + s_j^2\,X^{-\tilde{a}_j} = (1 - s_j^1 - s_j^1)\,X^{0\cdot\tilde{a}_j} + s_j^1\,X^{1\cdot\tilde{a}_j} + s_j^2\,X^{(-1)\cdot\tilde{a}_j}$.

As in Section 3.1, we define the $2n$ bootstrapping keys $\mathsf{bsk}[2(j-1)+i] \leftarrow \mathrm{GGSW}_{\mathfrak{z}'}(s_j^i)$ for all $1 \le j \le n$ and for all $1 \le i \le 2$. An application of the GGSW encryption to the above relation (1) leads to

$$\mathrm{GGSW}_{\mathfrak{z}'}(X^{s_j\,\tilde{a}_j}) \leftarrow (X^{\tilde{a}_j} - 1)\,\mathsf{bsk}[2j-1] + (X^{-\tilde{a}_j} - 1)\,\mathsf{bsk}[2j] + \mathrm{GGSW}_{\mathfrak{z}'}(1) \ .$$

We therefore obtain a new algorithm for the blind rotation. It is given in Algorithm 3.

Interestingly, the calculation of a blind rotation with ternary keys using Algorithm 3 only requires n external products.

Algorithm 3: Blind rotation (ternary case), revisited.

$\mathtt{acc} \leftarrow (0, \ldots, 0, X^{-\tilde{b}} \cdot \overline{v})$
for $j = 1$ **to** n **do**
  $\mathtt{acc} \leftarrow \mathtt{acc} + \left((X^{\tilde{a}_j} - 1)\,\mathsf{bsk}[2j-1] + (X^{-\tilde{a}_j} - 1)\,\mathsf{bsk}[2j]\right) \boxdot \mathtt{acc}$
end for
return acc

4 Extensions and Generalizations

4.1 Higher Radices

The proposed approach can be extended to support arbitrary formats of keys.

For full generality, we suppose that the keys are drawn from an arbitrary set $\mathcal{S}$ (e.g., $\mathcal{S} = \{0, 1, -1\}$ for ternary keys). We let $m = \#\mathcal{S}$ denote the cardinality of $\mathcal{S}$. The monomial $X^{s_j\,\tilde{a}_j}$ can be written additively as

$$X^{s_j\,\tilde{a}_j} = \sum_{t \in \mathcal{S}} \underbrace{\mathbb{1}\{t = s_j\}}_{:=\sigma_{j,t}} X^{t\,\tilde{a}_j} \quad . \tag{2}$$

In the above equation, $\mathbb{1}$ denotes the predicate function (*i.e.*, $\mathbb{1}\{t = s_j\} = 1$ when $t = s_j$ and $\mathbb{1}\{t = s_j\} = 0$ otherwise).

We define the set $I = \{0, \ldots, m-1\}$. We also define the "alphabet" vector $\boldsymbol{A} \in \mathcal{S}^m$ whose components are the different elements of $\mathcal{S}$. For example, back to ternary keys, we have $m = 3$ and write $\boldsymbol{A} = (0, 1, -1)$ so that $\boldsymbol{A}[0] = 0$, $\boldsymbol{A}[1] = 1$ and $\boldsymbol{A}[2] = -1$. Vector $\boldsymbol{A}$ gives rise to function

$$\mathsf{A} \colon I \to \mathcal{S}, i \mapsto \mathsf{A}(i) = \boldsymbol{A}[i]$$

taking on input an index $i \in I$ and returning the i^{th} component[2] of $\boldsymbol{A}$. Equation (2) can therefore be equivalently rewritten as

$$X^{s_j\,\tilde{a}_j} = \sum_{i=0}^{m-1} \mathbb{1}\{\mathsf{A}(i) = s_j\}\, X^{\mathsf{A}(i)\,\tilde{a}_j} \quad . \tag{3}$$

The first digit in the alphabet vector is $\tau_0 \coloneqq \boldsymbol{A}[0] = \mathsf{A}(0)$. As the predicate function $\mathbb{1}\{\mathsf{A}(i) = s_j\}$ is true for exactly one index $i \in \{0, \ldots, m-1\}$, we deduce from Eq. (3) that

$$X^{s_j\,\tilde{a}_j} = X^{\mathsf{A}(0)\,\tilde{a}_j} + \sum_{i=1}^{m-1} \mathbb{1}\{\mathsf{A}(i) = s_j\} \left(X^{\mathsf{A}(i)\,\tilde{a}_j} - X^{\mathsf{A}(0)\,\tilde{a}_j}\right) \quad . \tag{4}$$

Interestingly, compared to Eq. (3), the formulation of Eq. (4) involves only $m-1$ predicate evaluations.

[2] Starting at $i = 0$.

Hence, defining $n(m-1)$ bootstrapping keys $\mathsf{bsk}[(m-1)(j-1)+i] \leftarrow \mathrm{GGSW}_{\mathfrak{z}'}(\sigma_{j,t})$ with $\sigma_{j,t} = \mathbb{1}\{t = s_j\}$ and $t = \mathsf{A}(i)$, for all $1 \le j \le n$ and for all $1 \le i \le m-1$, we obtain Algorithm 4. Somewhat surprisingly, the number of external products remains equal to n. For a given security level, we note the value of n is a decreasing function of m; see Appendix A.

Algorithm 4: Blind rotation (higher-radix case).

1 $\mathsf{acc} \leftarrow (0, \ldots, 0, X^{-\tilde{b}} \cdot \overline{v})$
2 **for** $j = 1$ **to** n **do**
3 $\quad \mathsf{acc} \leftarrow X^{\mathsf{A}(0)\,\tilde{a}_j} \cdot \mathsf{acc} + \left(\sum_{i=1}^{m-1} (X^{\mathsf{A}(i)\,\tilde{a}_j} - X^{\mathsf{A}(0)\,\tilde{a}_j})\,\mathsf{bsk}[(m-1)(j-1)+i]\right) \boxdot \mathsf{acc}$
4 **end for**
5 **return** acc

Remark 2. If the digit alphabet $\mathcal{S}$ contains the digit 0, it is advantageous to set the first digit τ_0 to 0 (and thus $\mathsf{A}(0) = 0$) so that Line 3 in Algorithm 4 simplifies to

$$\mathsf{acc} \leftarrow \mathsf{acc} + \left(\sum_{i=1}^{m-1} (X^{\mathsf{A}(i)\,\tilde{a}_j} - 1)\,\mathsf{bsk}[(m-1)(j-1)+i]\right) \boxdot \mathsf{acc}$$

4.2 Multi-Digit Approach

The number of external products can be further decreased. We combine and extend the multi-bit approach of [17,4] to the case of a secret key expressed as a series of digits in a higher radix. The idea is to process several digits concurrently.

In the previous section, the monomial $X^{s_j\,\tilde{a}_j}$ is additively expressed under the form $X^{s_j\,\tilde{a}_j} = \sum_{t\in\mathcal{S}} \sigma_{j,t} \cdot X^{t\,\tilde{a}_j}$ with $\sigma_{j,t} = \mathbb{1}\{t = s_j\}$. Let $\tau_0 = \boldsymbol{A}[0] = \mathsf{A}(0)$ denote the first digit in the alphabet vector. With two digits, we can express the corresponding monomial as

$$\begin{aligned}
&X^{s_{j_1}\,\tilde{a}_{j_1} + s_{j_2}\,\tilde{a}_{j_2}} \\
&\quad = X^{s_{j_1}\,\tilde{a}_{j_1}}\, X^{s_{j_2}\,\tilde{a}_{j_2}} \\
&\quad = \left(\sum_{t\in\mathcal{S}} \sigma_{j_1,t} \cdot X^{t\,\tilde{a}_{j_1}}\right)\left(\sum_{t\in\mathcal{S}} \sigma_{j_2,t} \cdot X^{t\,\tilde{a}_{j_2}}\right) \\
&\quad = \sum_{t_1\in\mathcal{S}} \sum_{t_2\in\mathcal{S}} \sigma_{j_1,t_1}\sigma_{j_2,t_2} \cdot X^{t_1\,\tilde{a}_{j_1}} X^{t_2\,\tilde{a}_{j_2}} \\
&\quad = X^{\tau_0(\tilde{a}_{j_1}+\tilde{a}_{j_2})} + \sum_{(t_1,t_2)\in\mathcal{S}^2\setminus\{(\tau_0,\tau_0)\}} \sigma_{j_1,t_1}\sigma_{j_2,t_2} \cdot \left(X^{t_1\,\tilde{a}_{j_1} + t_2\,\tilde{a}_{j_2}} - X^{\tau_0(\tilde{a}_{j_1}+\tilde{a}_{j_2})}\right) .
\end{aligned}$$

More generally, for d digits, we get

$$X^{\sum_{l=1}^{d} s_{j_l}\,\tilde{a}_{j_l}} = X^{\tau_0 \sum_{l=1}^{d} \tilde{a}_{j_l}} + \sum_{(t_1,\ldots,t_d)\in\mathcal{S}^d\setminus\{(\tau_0,\ldots,\tau_0)\}} \left(\bigwedge_{l=1}^{d} \sigma_{j_l,t_l}\right) \cdot \left(X^{\sum_{l=1}^{d} t_l\,\tilde{a}_{j_l}} - X^{\tau_0 \sum_{l=1}^{d} \tilde{a}_{j_l}}\right) .$$

To ease the presentation, we henceforth assume that $d \mid n$. We can write $\sum_{j=1}^{n} s_j\,\tilde{a}_j = \sum_{h=1}^{n/d} \sum_{l=1}^{d} s_{(h-1)d+l}\,\tilde{a}_{(h-1)d+l}$. We define $\frac{n}{d}(m^d-1)$ bootstrapping keys

$$\mathsf{bsk}[(m^d-1)(h-1) + \textstyle\sum_{l=1}^{d} i_l\, m^{l-1}] \leftarrow \mathrm{GGSW}_{\mathfrak{z}'}\left(\bigwedge_{l=1}^{d} \sigma_{(h-1)d+l,t_l}\right)$$

with $(t_1,\ldots,t_d) = (\mathsf{A}(i_1),\ldots,\mathsf{A}(i_d))$, for all $1 \le h \le n/d$ and for all $(i_1,\ldots,i_d) \in \{0,\ldots,m-1\}^d$ with $(i_1,\ldots,i_d) \neq (0,\ldots,0)$. The number of external products decreases to n/d.

Algorithm 5: Blind rotation (multi-digit case).

```
1 acc ← (0, ..., 0, X^{-b̃} · v̄)
2 for h = 1 to n/d do
3 |  acc ← X^{A(0) Σ_{l=1}^{d} ã_{(h-1)d+l}} · acc + ( Σ_{0≤i_1,...,i_d≤m-1, (i_1,...,i_d)≠(0,...,0)} (X^{Σ_{l=1}^{d} A(i_l) ã_{(h-1)d+l}} −
  |    X^{A(0) Σ_{l=1}^{d} ã_{(h-1)d+l}}) bsk[(m^d − 1)(h − 1) + Σ_{l=1}^{d} i_l m^{l-1}]) ⊡ acc
4 end for
5 return acc
```

Remark 3. Again, similarly to Remark 2, when digit $0 \in \mathcal{S}$, setting $\boldsymbol{A}[0] = \mathsf{A}(0) = 0$, Line 3 in Algorithm 5 simplifies and becomes

$$\mathsf{acc} \leftarrow \mathsf{acc} + \Big(\sum_{\substack{0\le i_1,\ldots,i_d\le m-1\\ (i_1,\ldots,i_d)\neq(0,\ldots,0)}} (X^{\sum_{l=1}^{d} \mathsf{A}(i_l)\,\tilde{a}_{(h-1)d+l}} - 1)\,\mathsf{bsk}[(m^d-1)(h-1) + \textstyle\sum_{l=1}^{d} i_l\, m^{l-1}]\Big) \boxdot \mathsf{acc}$$

5 Performance Analysis and Experiments

In this section, we analyze the performance of our extended blind rotation. Our efficiency comparisons are based on a C library called `zlib` specifically designed to conduct experiments on TFHE. Lightweight and modular, the purpose of `zlib` is to finely tune the bootstrapping procedure, establish the performance profile of various algorithmic strategies on CPUs, and see how they compare depending on their parameters.

5.1 LWE Estimator for Security Estimates

The security of LWE encryption depends on its parameters in a way that is dictated by the current state-of-the-art attacks on LWE. We rely on the LWE Estimator [1][3] to enforce a desired security level. Given an LWE dimension n, a secret key uniformly drawn from $\mathcal{S}^n$ with $\#\mathcal{S} = m$, a ciphertext precision p and a noise variance v, this tool provides a security estimate

$$\lambda = \mathsf{LWE\text{-}security}(n, m, p, v)$$

against an IND-CPA adversary to whom an unbounded number of encryptions of zero are given. The LWE-security function is also valid for GLWE and GGSW ciphertexts when n is replaced with kN.

5.2 Nominal Setting

Our nominal setting, and basis for comparison, is set to $(d, m) = (1, 2)$. This means that the LWE secret key embedded in the bootstrapping key is a usual (non-sparse) binary key $s \in \{0, 1\}^n$ and each element of the bootstrapping key is a GGSW encryption of bit s_j, $1 \leq j \leq n$.

FFT-based external product. Each external product is performed as follows. The input accumulator is a GLWE ciphertext; *i.e.*, a vector of $k+1$ torus polynomials modulo $X^N + 1$ with coefficients of p_{acc} bits. We fix $k = 1$, $N = 1024$ and $p_{acc} = 64$. Each torus polynomial is decomposed into ℓ integer polynomials with coefficients in $\{-B/2, \ldots, B/2\}$ where we pick $\ell = 3$ and $B = 2^\beta = 2^8$. We then apply a radix-2 negacyclic FFT to every one of these integer polynomials, resulting in a matrix of $\ell(k+1)$ complex arrays in $\mathbb{C}^{N/2}$. We easily compute the tensor

$$T = (X^{\tilde{a}_j} - 1) \cdot \mathsf{bsk}[j]$$

on the fly since the FFT coefficients of $X^{\tilde{a}_j}$ are just cyclic powers of $e^{i\pi/N}$ and can be derived from the twiddle factors of the FFT that are already precomputed and stored. Applying a complex matrix-tensor product then gives us $k+1$ arrays in $\mathbb{C}^{N/2}$, which are converted back to torus polynomials in the standard domain by applying a radix-2 backward FFT and rounding. The $k+1$ polynomials are then added up to the accumulator modulo $2^{p_{acc}} = 2^{64}$, which gives the output value of the accumulator.

Running time. Instrumenting `zlib` with this setting allows us to measure the average number of clock cycles $\mathrm{time}^{\mathrm{XP}}(1, 2)$ required to perform one external product on a reference architecture. The total running time of the blind rotation is then

$$\mathrm{time}^{\mathrm{BR}}(1, 2) = n \cdot \mathrm{time}^{\mathrm{XP}}(1, 2)$$

where we fix $n = 640$. This particular setting for $n, N, k, \ell, \beta, p_{acc}$ complies with a parameter set often used in implementations of TFHE.

[3] Available at https://bitbucket.org/malb/lwe-estimator/src/master/.

Output variance. We neglect the approximation errors that are due to the use of 64-bit floating-point numbers in FFT conversions and operations in the FFT domain. One can show that one external product increases the variance of the accumulated noise by

$$\mathrm{var}^{\mathrm{XP}}(1,2) = (k+1) \cdot N \cdot \mathbb{M}_2(\ell, B) \cdot \mathrm{var}^{\mathrm{bsk}}$$

where the term

$$\mathbb{M}_2(\ell, B) = \ell \cdot \frac{(B+2)(B^2-B+1)}{12 \cdot (B+1)} + (1 - (-\tfrac{1}{B})^{\ell}) \cdot \frac{B^2}{4 \cdot (B+1)^2}$$

is the second statistical moment of the coefficients of the decomposed integer polynomials right before their conversion to the FFT domain. Finally, $\mathrm{var}^{\mathrm{bsk}}$ is the noise variance of the GGSW ciphertexts that compose the bootstrapping key. We fix the overall security level to $\lambda = 128$ and find $\mathrm{var}^{\mathrm{bsk}}$ by solving

$$\mathsf{LWE\text{-}security}(kN, p_{acc}, 2, \mathrm{var}^{\mathrm{bsk}}) = 128$$

which yields $\mathrm{var}^{\mathrm{bsk}} = 2^{-50.32}$. The output variance of the blind rotation is then

$$\mathrm{var}^{\mathrm{BR}}(1,2) = n \cdot \mathrm{var}^{\mathrm{XP}}(1,2) \ .$$

5.3 Extended Setting

Keeping the same parameters as in the nominal setting, we now generalize it to arbitrary pairs (d, m) with $d \geq 1$ and $m \geq 2$.

Extended external product. In comparison with the nominal setting, the only adaptation in the external product is that the tensor

$$T = (X^{\tilde{a}_j} - 1) \cdot \mathsf{bsk}[j]$$

is now generalized to

$$T = \sum_{\substack{0 \leq i_1, \ldots, i_d \leq m-1 \\ (i_1, \ldots, i_d) \neq (0, \ldots, 0)}} (X^{\sum_{l=1}^{d} \mathsf{A}(i_l)\, \tilde{a}_{(h-1)d+l}} - 1)\, \mathsf{bsk}[(m^d-1)(h-1) + \textstyle\sum_{l=1}^{d} i_l\, m^{l-1}]$$

which requires $m^d - 1$ operations of the form $(X^{\alpha} - 1) \cdot \mathsf{bsk}[j]$ instead of just one.

Running time. We measure the average number of clock cycles $\mathrm{time}^{\mathrm{XP}}(d, m)$ using `zlib` and find that

$$\mathrm{time}^{\mathrm{XP}}(d, m) = \big(1 + \big(m^d - 2\big) \cdot \Delta\big) \cdot \mathrm{time}^{\mathrm{XP}}(1, 2)$$

with $\Delta \approx 0.1557$.

Output variance. The noise variance added by an external product is now

$$\mathrm{var}^{\mathrm{XP}}(d,m) = (k+1)\cdot N\cdot \mathbb{M}_2(\ell,B)\cdot\left(m^d-1\right)\cdot \mathrm{var}^{\mathrm{bsk}} = \left(m^d-1\right)\cdot \mathrm{var}^{\mathrm{XP}}(1,2)$$

and the output variance of the blind rotation is

$$\mathrm{var}^{\mathrm{BR}}(d,m) = \frac{n}{d}\cdot \mathrm{var}^{\mathrm{XP}}(d,m)$$

However, the value of n can now be decreased slightly due to $m \geq 2$. Indeed, $n = 640$ was chosen to verify

$$\mathsf{LWE\text{-}security}(n,2,64,v) = 128$$

for a certain noise variance v, whereas we now require

$$\mathsf{LWE\text{-}security}(n,m,64,v) = 128$$

for the same variance v in our extended setting. Based on the data-points collected from LWE Estimator and given in Appendix A, we experimentally find replacement values for $n = n(m)$ as shown on Table 1.

Table 1. Optimal values of n as a function of $m \in \{2,\ldots,10\}$ for 128-bit security.

m	2	3	4	5	6	7	8	9	10
$n(m)$	640	610	591	579	569	561	555	549	544

5.4 Finding Optimal Settings

Putting it all together, our generalization gives the following ratios:

$$\frac{\mathrm{var}^{\mathrm{BR}}(d,m)}{\mathrm{var}^{\mathrm{BR}}(1,2)} = \frac{n(m)}{n(2)}\cdot\frac{m^d-1}{d}\,, \tag{5}$$

$$\frac{\mathrm{time}^{\mathrm{BR}}(d,m)}{\mathrm{time}^{\mathrm{BR}}(1,2)} = \frac{n(m)}{n(2)}\cdot\frac{1+\left(m^d-2\right)\cdot\Delta}{d}\,, \qquad \Delta \approx 0.1557 \tag{6}$$

$$\frac{\mathrm{keysize}^{\mathrm{BR}}(d,m)}{\mathrm{keysize}^{\mathrm{BR}}(1,2)} = \frac{n(m)}{n(2)}\cdot\frac{m^d-1}{d}\,. \tag{7}$$

We see that the ratios (5) and (7) are identical and we denote them by 2^u, while the ratio (6) is denoted 2^v. A given (u,v) pair indicates that a degradation of the output variance and key size by a factor 2^u yields a speedup factor of 2^{-v}. What we are after is to find the values of (d,m) that provide the most interesting (u,v) pairs, namely, ones where both u and v are minimized.

Due to the nature of these formulas, we easily see that no optimum exists for (d, m) that would simultaneously minimize the key size on one hand and the running time on the other. Instead, we take the set S of pairs (u, v) derived from all the possible settings (d, m) where $d \in \{1, \ldots, 10\}$ and $m \in \{2, \ldots, 10\}$, and eliminate from S the dominated points; *i.e.*, pairs $(u_2, v_2) \in S$ such that there exists $(u_1, v_1) \in S$ verifying

$$(u_1 \leq u_2 \text{ and } v_1 < v_2) \text{ or } (u_1 < u_2 \text{ and } v_1 \leq v_2) \ .$$

Dominated points can be safely eliminated since they are strictly less efficient than other reachable points. We end up with a subset $S_{\text{best}} \subseteq S$ of non-dominated points known as the Pareto front of S. The elements of S_{best} identify the best possible trade-offs among all possible settings, and are displayed on Figure 1.

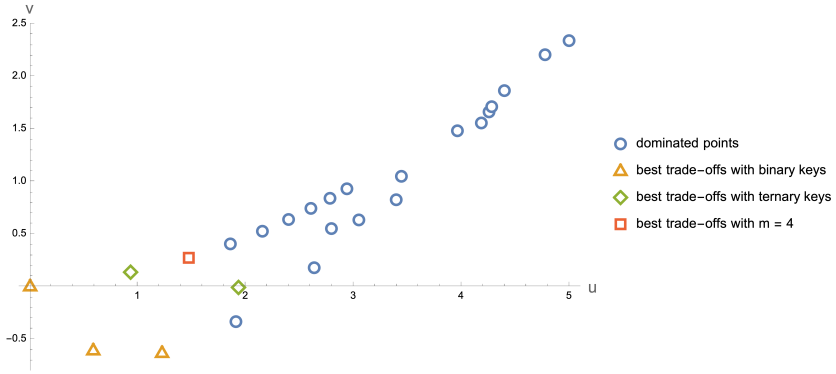

Fig. 1. Points (u, v) obtained from screening m through $\{2, \ldots, 10\}$ and d through $\{1, \ldots, 10\}$, and their Pareto front (orange triangles). Other points exist outside of the displayed range but they are all dominated. The nominal parameters of the blind rotation are $N = 1024$, $k = 1$, $\ell = 3$, $\beta = 8$ and $n = 640$.

To conclude our experiments, we see that in addition to the nominal setting that we have chosen ($u = v = 0$), two other trade-off points of interest appear:

- $\boldsymbol{(d, m) = (2, 2)}$, for which a 50% increase in key size and output variance increases speed by 52.5%, and
- $\boldsymbol{(d, m) = (3, 2)}$, for which a 133% increase in key size and output variance increases speed by 55.1%.

Should one move away from binary keys for security reasons, the next best trade-off points become

- $\boldsymbol{(d, m) = (1, 3)}$, for which a 91% increase in key size and output variance decreases speed by 9.22%, and
- $\boldsymbol{(d, m) = (2, 3)}$, for which a 281% increase in key size and output variance increases speed by 0.4%.

This assumes that the security estimates for ternary keys and $m \geq 4$ are left untouched by an alleged attack against binary keys.

6 Conclusion

This paper adapted the blind rotation as used in the bootstrapping in FHEW or TFHE to support ternary keys. The resulting implementation is about twice faster than a previous method by Micciancio and Polyakov and with the same memory requirements for the bootstrapping keys as in their method. We also extended the proposed approach to higher radices and generalized it by processing several digits at a time—at the expense of further memory requirements. An analysis of the various trade-offs provided by the choice of the radix and of the number of processed digits was provided. In particular, useful trade-offs include processing 2 or 3 bits concurrently for binary keys and 1 or 2 digits for ternary keys.

Acknowledgements. We are grateful to Ben Curtis for his help in compiling Tables 2 and 3. We are also grateful to the anonymous referees for useful comments.

References

1. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. doi:10.1515/jmc-2015-0016.
2. Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasilinear time. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013. doi:10.1007/978-3-642-40041-4_1.
3. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314. Springer, 2014. doi:10.1007/978-3-662-44371-2_17.
4. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 483–512. Springer, 2018. doi:10.1007/978-3-319-96878-0_17.
5. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13:1–13:36, 2014. Earlier version in ITCS 2012. doi:10.1145/2633600.
6. Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In M. Naor, editor, *5th Innovations in Theoretical Computer Science (ITCS 2014)*, pages 1–12. ACM Press, 2014. doi:10.1145/2554797.2554799.

7. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. doi:10.1007/s00145-019-09319-x.
8. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In S. Dolev et al., editors, *Cyber Security Cryptography and Machine Learning (CSCML 2021)*, volume 12716 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2021. doi:10.1007/978-3-030-78086-9_1.
9. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015. doi:10.1007/978-3-662-46800-5_24.
10. Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In J.-S. Coron and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 528–558. Springer, 2016. doi:10.1007/978-3-662-49896-5_19.
11. Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010. Earlier version in STOC 2009. doi:10.1145/1666420.1666444.
12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In M. Fischlin et al., editors, *Public Key Cryptography – PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012. doi:10.1007/978-3-642-30057-8_1.
13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013. doi:10.1007/978-3-642-40041-4_5.
14. Marc Joye. Guide to fully homomorphic encryption over the [discretized] torus. Cryptology ePrint Archive, Report 2021/1402, 2021. https://ia.cr/2021/1402.
15. Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In M. Brenner et al., editors, *9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2021)*, pages 17–28. ACM Press, 2021. doi:10.1145/3474366.3486924.
16. Ronald L. Rivest, Len Adleman, and Michael L. Detouzos. On data banks and privacy homomorphisms. In R. A. DeMillo et al., editors, *Foundations of Secure Computation*, pages 165–179. Academic Press, 1978. Available at https://people.csail.mit.edu/rivest/pubs.html#RAD78.
17. Tanping Zhou, Xiaoyuan Yang, Longfei Liu, Wei Zhang, and Ningbo Li. Faster bootstrapping with multiple addends. *IEEE Access*, 6:49868–49876, 2018. doi:10.1109/ACCESS.2018.2867655.

A Tables

Although for higher radices the number of external products remains equal to n, the value of n is a decreasing function of m. Playing with LWE Estimator, at a security level of 128 bits, we get the following tables.

Table 2. LWE dimension n as a function of m for different values for the noise standard deviation σ, for $q = 2^{32}$

σ	m								
	2	3	4	5	6	7	8	9	10
−30	1208	1176	1160	1152	1136	1128	1120	1120	1112
−29	1168	1144	1120	1112	1104	1088	1088	1080	1072
−28	1136	1104	1088	1072	1064	1056	1048	1040	1040
−27	1096	1064	1048	1032	1024	1016	1008	1000	1000
−26	1056	1024	1008	1000	984	976	968	968	960
−25	1016	992	968	960	952	944	936	928	920
−24	976	952	936	920	912	904	896	888	880
−23	944	912	896	880	872	864	856	848	848
−22	904	872	856	848	832	824	816	816	808
−21	864	840	816	808	800	784	784	776	768
−20	824	800	784	768	760	752	744	736	728
−19	792	760	744	728	720	712	704	696	696
−18	752	720	704	688	680	672	664	664	656
−17	712	680	664	656	640	632	632	624	616
−16	672	648	624	616	608	600	592	584	576
−15	640	608	592	576	568	560	552	544	544
−14	600	568	552	536	528	520	512	512	504
−13	560	528	512	496	488	480	472	472	464
−12	528	496	472	464	456	448	440	432	424
−11	488	456	432	424	416	408	400	392	392
−10	448	416	400	384	376	368	360	352	352
−9	408	376	360	344	336	328	320	320	312
−8	376	336	320	312	296	288	288	280	272
−7	336	304	280	272	256	256	248	240	240
−6	296	264	240	232	224	216	208	208	200

Table 3. LWE dimension n as a function of m for different values for the noise standard deviation σ, for $q = 2^{64}$

σ	m								
	2	3	4	5	6	7	8	9	10
−62	2424	2392	2376	2368	2352	2344	2336	2336	2328
−61	2384	2360	2336	2328	2320	2304	2304	2296	2288
−60	2344	2320	2304	2288	2280	2272	2264	2256	2248
−59	2304	2280	2264	2248	2240	2232	2224	2216	2208
−58	2272	2240	2224	2208	2200	2192	2184	2176	2176
−57	2232	2200	2184	2176	2160	2152	2144	2144	2136
−56	2192	2168	2152	2136	2128	2120	2112	2104	2096
−55	2152	2128	2112	2096	2088	2080	2072	2064	2056
−54	2120	2088	2072	2056	2048	2040	2032	2024	2024
−53	2080	2056	2032	2024	2016	2000	2000	1992	1984
−52	2048	2016	2000	1984	1976	1968	1960	1952	1952
−51	2008	1976	1960	1952	1936	1928	1920	1912	1912
−50	1968	1944	1920	1912	1896	1888	1880	1880	1872
−49	1928	1904	1888	1872	1864	1856	1848	1840	1832
−48	1888	1864	1848	1832	1824	1816	1808	1800	1792
−47	1856	1824	1808	1792	1784	1776	1768	1760	1760
−46	1816	1784	1768	1760	1744	1736	1728	1728	1720
−45	1776	1752	1728	1720	1712	1696	1696	1688	1680
−44	1736	1712	1696	1680	1672	1664	1656	1648	1640
−43	1704	1672	1656	1640	1632	1624	1616	1608	1608
−42	1664	1640	1616	1608	1600	1592	1584	1576	1568
−41	1624	1600	1584	1568	1560	1552	1544	1536	1528
−40	1592	1560	1544	1528	1520	1512	1504	1496	1496
−39	1552	1520	1504	1488	1480	1472	1464	1456	1456
−38	1512	1480	1464	1456	1440	1432	1424	1424	1416
−37	1472	1448	1424	1416	1408	1400	1392	1384	1376
−36	1432	1408	1392	1376	1368	1360	1352	1344	1336
−35	1400	1368	1352	1336	1328	1320	1312	1304	1304
−34	1360	1336	1312	1304	1296	1280	1280	1272	1264
−33	1320	1296	1280	1264	1256	1248	1240	1232	1224
−32	1288	1256	1240	1224	1216	1208	1200	1192	1192
−31	1248	1216	1200	1184	1176	1168	1160	1152	1152
−30	1208	1176	1160	1152	1136	1128	1120	1120	1112
−29	1168	1144	1120	1112	1104	1088	1088	1080	1072
−28	1136	1104	1088	1072	1064	1056	1048	1040	1040
−27	1096	1064	1048	1032	1024	1016	1008	1000	1000

−26	1056	1024	1008	1000	984	976	968	968	960
−25	1016	992	968	960	952	944	936	928	920
−24	976	952	936	920	912	904	896	888	880
−23	944	912	896	880	872	864	856	848	848
−22	904	872	856	848	832	824	816	816	808
−21	864	840	816	808	800	784	784	776	768
−20	824	800	784	768	760	752	744	736	728
−19	792	760	744	728	720	712	704	696	696
−18	752	720	704	688	680	672	664	664	656
−17	712	680	664	656	640	632	632	624	616
−16	672	648	624	616	608	600	592	584	576
−15	640	608	592	576	568	560	552	544	544
−14	600	568	552	536	528	520	512	512	504
−13	560	528	512	496	488	480	472	472	464
−12	528	496	472	464	456	448	440	432	424
−11	488	456	432	424	416	408	400	392	392
−10	448	416	400	384	376	368	360	352	352
−9	408	376	360	344	336	328	320	320	312
−8	376	336	320	312	296	288	288	280	272
−7	336	304	280	272	256	256	248	240	240
−6	296	264	240	232	224	216	208	208	200