Finding and Evaluating Parameters for BGV

Johannes Mono[1], Chiara Marcolla[2], Georg Land[1,3],
Tim Güneysu[1,3], and Najwa Aaraj[2]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Germany
[2] Technology Innovation Institute, Abu Dhabi, United Arab Emirates
[3] DFKI GmbH, Cyber-Physical Systems, Bremen, Germany

Abstract. The BGV scheme is a state-of-the-art fully homomorphic encryption (FHE) scheme. Encryption is based on the Learning with Errors over rings (RLWE) assumption and thus each ciphertext has an associated error that grows with each homomorphic operation. To avoid failure during decryption, the growing error, also called critical quantity, needs to stay below a certain threshold. This requires a trade-off between security and error margin that influences the parameters specific to each use case. Choosing such parameters, for example the polynomial degree or the ciphertext modulus, is a challenge and requires expert knowledge.

The main idea of our work is to improve the current state of BGV parameter selection. More specifically, we provide a parameter generator for the leveled BGV scheme using theoretical bounds on the error growth and an empirically derived formula for the security estimate. For the former, we combine previous analysis using the canonical embedding norm and analysis of the residue number system. For the latter, we develop a model based on data from the Lattice Estimator tool and coupled optimization. Finally, we provide the open-source generator which outputs easy-to-use code snippets for the BGV libraries HElib and PALISADE.

Keywords: Fully Homomorphic Encryption, BGV, Parameter Generation, HElib, PALISADE

1 Introduction

Since Gentry's seminal work in 2009 [10], fully homomorphic encryption (FHE) has attracted a lot of attention from the cryptographic research community [1,18,19]. FHE enables arbitrary computations on encrypted data and opens up new possibilities in data processing. As an example, hospitals analyzing health information can work only on encrypted data and provide clients with an encrypted result, thus not risking to leak any sensitive data.

The Brakerski Gentry Vaikuntanathan (BGV) scheme [4] is currently considered to be one of the state-of-the-art FHE schemes. BGV is based on the Learning with Errors (LWE) problem [23] and its ring variant RLWE [17]. RLWE-based FHE schemes, including BGV, need to keep the error associated with each ciphertext below a certain threshold as decryption would fail otherwise. This requires

a trade-off between security (small ciphertext modulus) and error margin (big ciphertext modulus).

In general, choosing parameters such as the polynomial degree d or the ciphertext modulus q is a challenge with FHE schemes and requires expert knowledge specific to each scheme. The FHE community took a first step towards standardization with the Homomorphic Encryption Security Standard [2]. The standard provides parameter sets based on the LWE Estimator, a software tool to determine the security level of RLWE instances[4] More specifically, they provide upper limits on the size of the ciphertext modulus for certain security levels λ and polynomial degrees d in the form of lookup tables.

However, there are other parameters to consider and users and developers alike have to make additional choices with state-of-the-art software libraries. A real-world example is the programming interface of HElib [13,26], an open source FHE library by Halevi and Shoup that provides a software implementation of BGV. The following is a stripped-down setup routine in an example shipped with HElib (Listing 1.1).

```
auto ctx = ContextBuilder<BGV>()
    .m(32109)  // cyclic order
    .p(4999)   // plaintext modulus
    .r(1)      // Hensel lifting
    .bits(500) // length of modulus chain
    .c(2)      // key switching columns
    .build()
```

Listing 1.1. BGV setup routine in HElib

A user needs to choose five parameters in order to use the BGV implementation. For researchers familiar with FHE, this flexibility is valuable. For other users however, this burden of choice increases the difficulty of simply using the library securely.

With PALISADE [21], another open source FHE library that also implements BGV, even more parameters have to be selected in the BGV example:

```
auto ctx = CryptoContext<DCRTPoly>::BGVrns(
    2,                  // cyclic order
    65537,              // plaintext modulus
    HEStd_128_classic,  // security level
    3.2,                // error standard deviation
    2,                  // multiplicative depth
    OPTIMIZED,          // secret distribution
    BV);                // key switching method
```

Listing 1.2. BGV setup routine in PALISADE

Thus, a user needs to choose a polynomial implementation as well as seven additional parameters (Listing 1.2).

[4] Its successor is the Lattice Estimator [3]
`https://github.com/malb/lattice-estimator`

Taking a look at other setup options in the code base, PALISADE also provides a simple setup routine by setting most parameters to specific defaults. The trade-off however is a non-optimized error margin leading to bigger-than-necessary ciphertext and key sizes as well as slower execution times. Although not perfect, we want to remark that both libraries manage to reduce the complexity of choosing parameters from a user's perspective.

The main idea of our work is to improve the current usability of the BGV scheme with the following contributions:

- We empirically derive formulas for parameters generation using the lattice estimator [3]. Specifically, for a given ciphertext modulus, we provide the the polynomial degree as input and receive the security level as output (Section 6).
- We provide an easy-to-use, interactive parameter generator for the leveled BGV scheme using our theoretical and empirical formulas (Section 5) following previous works [2, 6, 7, 11, 15]. To the best of our knowledge, we provide the first use-case focused generator for the BGV scheme.
- The generator outputs code snippets with example circuits for either PALISADE or HElib (Section 7) as well as benchmarking code for the specific parameter set.
- We give a complete overview of the leveled BGV scheme that combines the residue number system (RNS) approach of Kim *et al.* [16] with the error analysis of Gentry *et al.* [11] and Costache *et al.* [6,7] (Section 3). In addition, we extend previous parameter analysis to include rotations in our model.

2 Preliminaries

2.1 Notations

For a positive integer m, we denote by $\mathbb{Z}_m$ the ring of integers modulo m. We denote by $\mathbb{Z}_m^* = \{x \in \mathbb{Z}_m \,|\, (x, m) = 1\}$ the multiplicative group of units. We denote by $R = \mathbb{Z}[x]/\langle \Phi_m(x) \rangle$ and by $R_p = \mathbb{Z}_p[x]/\langle \Phi_m(x) \rangle$, where p is an integer and $\Phi_m(x)$ is the cyclotomic polynomial (see Section 2.2). We denote by t and q the plaintext and the ciphertext modulus, respectively, and R_t the plaintext space. Moreover, we set $t \equiv 1 \mod m$ (for CRT - see Section 4) and q a chain of primes, such that

$$q = q_{L-1} = \prod_{j=0}^{L-1} p_j,$$

where p_i are roughly of the same size and $p_i \equiv 1 \mod m$ [11]. Moreover, for the scaling procedure we set $p_i \equiv 1 \mod t$ [6]. Note that L primes provide the multiplicative depth of the circuit, which is $L - 1$. So, for any level ℓ, we have $q_\ell = \prod_{j=0}^{\ell} p_j$.

Polynomials are denoted by lower letters such as a, vectors of polynomials are denoted in bold $\mathbf{a}$ and polynomial multiplication is denoted as $a \cdot b$ while multiplication with a scalar t is denoted as ta.

2.2 Mathematical Background

Cyclotomic polynomial Let $\mathbb{F}$ be a field and m be a positive integer. We recall that a m-th *root of unity* is a number $\zeta \in \mathbb{F}$ satisfying the equation $\zeta^m = 1$. It is called *primitive* if m is the smallest positive integer for which $\zeta^m = 1$. The m-th *cyclotomic polynomial* is defined as $\Phi_m(x) = \prod_{(j,m)=1}(x - \zeta^j)$. The degree of Φ_m is $\phi(m) = m\prod_{p|m}(1 - 1/p) = |\mathbb{Z}_m^*|$, Euler's totient function.

Canonical embedding and canonical norm In this section we recall the result of [6, 7, 15]. Let $a \in R$ be a polynomial. The *canonical embedding* of a is the vector obtained by evaluating a at all primitive m-th roots of unity. The *canonical embedding norm* of $a \in R$ is defined as $||a||^{can} = \max_{j\in\mathbb{Z}_m^*} |a(\zeta^j)|$. For a vector of polynomials $\mathbf{a} = (a_0, \ldots, a_{n-1}) \in R^n$, the canonical embedding norm is defined as $||\mathbf{a}||^{can} = \max_i ||a_i||^{can}$. For any polynomial $a, b \in R$, the following properties hold:

- $||a||^{can} \leq \phi(m)||a||_\infty$.
- $||ab||^{can} \leq ||a||^{can}||b||^{can}$.
- $||a||_\infty \leq c_m||a||^{can}$ for the *ring expansion factor* c_m.

Note that $c_m = 1$ if $\Phi_m(x)$ is a power-of-two [8].

Let us consider a random $a \in R$ where each coefficient is sampled independently from on of the following zero-mean distributions:

- $\mathcal{DG}_q(\sigma^2)$, the discrete Gaussian distribution with standard deviation σ over the interval $(-q/2, q/2]$.
- $\mathcal{DB}_q(\sigma^2)$, the discrete Binomial distribution with standard deviation σ over the interval $(-q/2, q/2]$.
- $\mathcal{U}_3$, the uniform distribution over the ternary set $\{\pm 1, 0\}$.
- $\mathcal{U}_q$, the uniform distribution over $\mathbb{Z}_q$.
- $\mathcal{ZO}(\rho)$, a distribution over the ternary set $\{0, \pm 1\}$ with probability $\rho/2$ for ± 1 and probability $1 - \rho$ for 0 with $\rho \in [0, 1]$.

If we choose $a \in R$ from the distributions above, the random variable $a(\zeta)$ has variance $V = \phi(m) \cdot V_a$, where V_a is the variance of each coefficient in a and it is bounded

$$||a||^{can} \leq D\sigma\sqrt{\phi(m)} = D\sqrt{\phi(m) \cdot V_a}, \tag{1}$$

for some D [6]. Moreover, the probability that the variable a exceeds its standard deviation by more than a factor of D is roughly $\mathrm{erfc}(D)$. Thus, we have to choose D large enough to obtain a reasonable failure probability. Specifically, $\mathrm{erfc}(6) \approx 2^{-55}$, $\mathrm{erfc}(5) \approx 2^{-40}$ and $\mathrm{erfc}(4.5) \approx 2^{-32}$.

If $a, b \in R$ are chosen randomly and γ is a constant, the following holds for the variances [7]:

- $V_{a+b} = V_a + V_b$.
- $V_{\gamma a} = \gamma^2 V_a$.
- $V_{ab} = \phi(m) V_a V_b$.

Thus, to study the variance of $||a||^{can}$, we have to study the variance V_a of each coefficient a_i of a. Specifically,

$$\begin{array}{llll} a_i \in \mathcal{U}_q & \Rightarrow V_a \approx q^2/12, & a_i \in \mathcal{U}_3 & \Rightarrow V_a = 2/3, \\ a_i \in \mathcal{DG}_q(\sigma^2) & \Rightarrow V_a = \sigma^2, & a_i \in \mathcal{ZO}(\rho) & \Rightarrow V_a = \rho. \end{array} \tag{2}$$

As in [6], we assume that messages behave as if selected uniformly at random from $\mathcal{U}_t$. Thanks to Equations (1) and (2), we have that

$$||m||^{can} \leq Dt\sqrt{\phi(m)/12}. \tag{3}$$

Lattices and Hermite Factor Let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$ be linearly independent vectors in $\mathcal{R}^n$, then the *lattice* $\mathcal{L}(B)$ generated by the *base* B is defined by

$$\mathcal{L} = \mathcal{L}(B) = \Big\{ \sum_{i=1}^{k} \gamma_i \mathbf{b}_i : \quad \gamma_i \in \mathbb{Z}, \mathbf{b}_i \in B \Big\}.$$

The dimension k of a lattice $\mathcal{L} \subset \mathcal{R}^n$ is called *rank*. The *volume* (or *determinant*) of $\mathcal{L}$ is defined as $\mathsf{Vol}(\mathcal{L}) = \sqrt{\det(B^t B)}$. In the special case that $\mathcal{L}$ is a full rank lattice, i.e. when $k = n$, we have that $\mathsf{Vol}(\mathcal{L}) = |\det(B)|$. Finally, we can define the *Hermite factor* δ_0^k as

$$\delta_0^k = ||\mathbf{b}_1||/\mathsf{Vol}(\mathcal{L})^{1/k} \tag{4}$$

where $\mathbf{b}_1$ is the shortest vector in the reduced base B of the lattice $\mathcal{L}$. The factor δ_0 is called the *root Hermite factor*.

3 The BGV Scheme

The BGV scheme provides multiple algorithms for encryption and decryption, for homomorphic operations and for controlling the error growth. In the following, we provide a definition of all algorithms except bootstrapping, we consider it out of scope for this work.

3.1 Key Generation, Encryption & Decryption

KeyGen(λ)
Define parameters and distributions with respect to λ. Sample $s \leftarrow \chi_s$, $a \leftarrow \mathcal{U}_{q_L}$ and $e \leftarrow \chi_e$. Output $\mathsf{sk} = s$ and $\mathsf{pk} = (b, a) \equiv (-a \cdot s + te, a) \pmod{q_L}$.

Enc$_{\mathsf{pk}}(m)$
Receive plaintext $m \in \mathcal{R}_t$ for $\mathsf{pk} = (b, a)$. Sample $u \leftarrow \chi_s$ and $e_0, e_1 \leftarrow \chi_e$.
Output $\mathfrak{c} = (\mathbf{c}, L, \nu_{\mathsf{clean}})$ with $\mathbf{c} = (c_0, c_1) \equiv (b \cdot u + te_0 + m, a \cdot u + te_1) \pmod{q_L}$.

$\mathrm{Dec}_{\mathsf{sk}}(\mathfrak{c})$

Receive extended ciphertext $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ for $\mathsf{sk} = s$. Decrypt with $c_0 + c_1 \cdot s \equiv m + te \pmod{q_\ell}$ and output $m \equiv m + te \bmod t$.

To understand the error growth and thus analyze the critical quantity ν for each *extended ciphertext* $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$, we apply the decryption algorithm. The following shows the decryption of a ciphertext after an encryption:

$$\begin{aligned} c_0 + c_1 \cdot s &\equiv (-a \cdot s + te) \cdot u + te_0 + m + (a \cdot u + te_1) \cdot s && \pmod{q_L} \\ &\equiv m + t(e \cdot u + e_1 \cdot s + e_0) && \pmod{q_L}. \end{aligned}$$

The critical quantity is thus defined as $[c_0 + c_1 \cdot s]_{q_\ell}$ for the associated level ℓ.

In general, decryption is correct as long as the error does not wrap around the modulus q_ℓ, that is $||\nu||_\infty \leq c_m ||\nu||^{can} < q_\ell/2$. Note that applying decryption is equivalent to evaluating the ciphertext $\mathbf{c}$ as polynomial in s, that is $c_0 + c_1 \cdot s \equiv \nu \bmod q_\ell$. In the following, we will often use this polynomial representation of a ciphertext to proof correctness of an algorithm or operation.

We derive the bounds for each operation using the canonical embedding norm (Section 2.2). For the encryption operation, we use Equations (2) and (3)

$$\begin{aligned} ||[c_0 + c_1 \cdot s]_{q_\ell}||^{can} &\leq D\sqrt{\phi(m) V_{[c_0 + c_1 \cdot s]_{q_\ell}}} = D\sqrt{\phi(m)\left(V_m + t^2 V_{e \cdot u + e_1 \cdot s + e_0}\right)} \\ &\leq D\sqrt{\phi(m)\left(V_m + t^2(\phi(m) V_e V_u + \phi(m) V_{e_1} V_s + V_{e_0})\right)}. \end{aligned}$$

Namely,

$$\mathsf{B}_{\mathsf{clean}} = Dt\sqrt{\phi(m)\left(1/12 + 2\phi(m) V_e V_s + V_e\right)}. \tag{5}$$

3.2 Addition, Multiplication & Constant Multiplication

$\mathrm{Add}(\mathfrak{c}, \mathfrak{c}')$

Receive extended ciphertexts $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and $\mathfrak{c}' = (\mathbf{c}', \ell, \nu')$.
Output $(\mathbf{c} + \mathbf{c}', \ell, \nu_{\mathsf{add}})$.

$\mathrm{Mul}(\mathfrak{c}, \mathfrak{c}')$

Receive extended ciphertexts $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and $\mathfrak{c}' = (\mathbf{c}', \ell, \nu')$.
Output $((c_0 \cdot c'_0, c_0 \cdot c'_1 + c_1 \cdot c'_0, c_1 \cdot c'_1), \ell, \nu_{\mathsf{mul}})$.

$\mathrm{MulConst}(\alpha, \mathfrak{c})$

Receive constant polynomial $\alpha \in \mathcal{R}_t$ and extended ciphertext $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$.
Output $(\alpha \cdot \mathbf{c}, \ell, \nu_{\mathsf{const}})$.

As long as the bound on each critical quantity stays below the decryption threshold, correctness follows with

$$\begin{aligned}
&\nu_{\mathsf{add}} = \nu + \nu' = [c_0 + c_1 \cdot s]_{q_\ell} + [c'_0 + c'_1 \cdot s]_{q_\ell} \equiv m + m' \bmod t \\
&\Rightarrow ||\nu_{\mathsf{add}}||^{can} \leq ||\nu||^{can} + ||\nu'||^{can} \\
&\nu_{\mathsf{mul}} = \nu \cdot \nu' = [c_0 + c_1 \cdot s]_{q_\ell} \cdot [c'_0 + c'_1 \cdot s]_{q_\ell} \equiv m \cdot m' \bmod t \\
&\Rightarrow ||\nu_{\mathsf{mul}}||^{can} \leq ||\nu||^{can} ||\nu'||^{can} \\
&\nu_{\mathsf{const}} = \alpha \cdot \nu = \alpha \cdot [c_0 + c_1 \cdot s]_{q_\ell} \equiv \alpha \cdot m \bmod t \\
&\Rightarrow ||\nu_{\mathsf{const}}||^{can} \leq ||\alpha||^{can} ||\nu||^{can} = Dt\sqrt{\tfrac{\phi(m)}{12}} ||\nu||^{can}.
\end{aligned}$$

Here, we also consider the constant α to be uniformly distributed in $\mathcal{R}_t$. Note that the output of the multiplication is still a polynomial in s, but of degree 2. We will later define key switching (see Section 3.4) to modify a ciphertext polynomial $c_0 + c_1 \cdot s + c_2 \cdot s^2$ back to another polynomial $c'_0 + c'_1 \cdot s$ encrypting the same plaintext.

3.3 Modulus Switching

Until now, the associated level for each ciphertext stayed the same in each operation. Modulus switching reduces the associated level and the critical quantity for a ciphertext, enabling leveled homomorphic computations. We consider each modulus switching as a *level boundary* and a maximum circuit depth L, which usually corresponds to the amount of multiplications M and one or two additional modulus switching operations.

The idea is to switch from a ciphertext modulus q_ℓ to a ciphertext modulus $q_{\ell'} = q_{\ell-1}$. We thus multiply the ciphertext by $\frac{q_{\ell'}}{q_\ell} = \frac{1}{p_\ell}$, roughly reducing the error by the same factor. But, as we need to output a valid ciphertext, we add a small correction term that (i) only influences the error, that is being a multiple of t, and (ii) modifies the ciphertext to be divisible by p_ℓ.

ModSwitch($\mathfrak{c}, \ell'$)

Receive extended ciphertext $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and target level $\ell' = \ell - 1$.
Set $\boldsymbol{\delta} = t[-\mathbf{c}t^{-1}]_{p_\ell}$ with $\boldsymbol{\delta} \equiv 0 \bmod t$ and $\boldsymbol{\delta} \equiv -\mathbf{c} \bmod p_\ell$ and

$$\mathbf{c}' = \frac{1}{p_\ell}(\mathbf{c} + \boldsymbol{\delta}) \pmod{q_{\ell'}}$$

and output $(\mathbf{c}', \ell', \nu_{\mathsf{ms}})$.

First, we want to show the correctness of modulus switching. Let $[c_0 + c_1 \cdot s]_{q_\ell} = c_0 + c_1 \cdot s - kq_\ell$ for some $k \in \mathbb{Z}$. For the same k, let

$$\begin{aligned}
[c'_0 + c'_1 \cdot s]_{q_{\ell'}} &= c'_0 + c'_1 \cdot s - kq_{\ell'} \\
&= \frac{1}{p_\ell}(c_0 + c_1 \cdot s + \delta_0 + \delta_1 \cdot s) - kq_{\ell'} \\
&= \frac{1}{p_\ell}([c_0 + c_1 \cdot s]_{q_\ell} + kq_\ell + \delta_0 + \delta_1 \cdot s) - kq_{\ell'} \\
&= \frac{1}{p_\ell}([c_0 + c_1 \cdot s]_{q_\ell} + \delta_0 + \delta_1 \cdot s) \\
&\equiv p_\ell^{-1} m \bmod t.
\end{aligned}$$

Note that we actually decrypt to the plaintext $p_\ell^{-1} m \bmod t$, but we can multiply a plaintext by p_ℓ either before encryption or after decryption. This issue does not exist for $p_\ell \equiv 1 \bmod t$, but finding such p_ℓ can be difficult in practice.

The error after the modulus switching is bounded by

$$||\nu_{\mathsf{ms}}||^{can} \equiv |||[c'_0 + c'_1 \cdot s]_{q_{\ell'}}||^{can} \leq \frac{1}{p_\ell}\left(||\nu||^{can} + ||\delta_0 + \delta_1 \cdot s||^{can}\right).$$

As $V_{\delta_i} = V_{tp_\ell} = \frac{t^2 p_\ell^2}{12}$, and thus $V_{\delta_0 + \delta_1 \cdot s} = \frac{t^2 p_\ell^2}{12}(1 + \phi(m)V_s)$, we have

$$||\nu_{\mathsf{ms}}||^{can} \leq \frac{1}{p_\ell}(||\nu||^{can} + D\sqrt{\phi(m)V_{\delta_0 + \delta_1 \cdot s}}) = \frac{1}{p_\ell}||\nu||^{can} + \mathsf{B}_{\mathsf{scale}},$$

with

$$\mathsf{B}_{\mathsf{scale}} = Dt\sqrt{\frac{\phi(m)}{12}(1 + \phi(m)V_s)}. \tag{6}$$

Note that, decryption is correct as long as $||\nu||^{can} < \frac{q_\ell}{2c_m} - p_\ell \mathsf{B}_{\mathsf{scale}}$ [10].

3.4 Key Switching

Key switching is used for (i) reducing the degree a ciphertext polynomial, usually the output of a multiplication, or (ii) changing the key after a rotation. For a multiplication, we convert the ciphertext term $c_2 \cdot s^2$ to a polynomial $c_0^{\mathsf{ks}} + c_1^{\mathsf{ks}} \cdot s$ and for a rotation, we convert the ciphertext term $c_1 \cdot \mathrm{rot}(s)$ to a polynomial $c_0^{\mathsf{ks}} + c_1^{\mathsf{ks}} \cdot s$. In the following, we will only analyze multiplication and more specifically, we will output $\mathbf{c}' = (c_0 + c_0^{\mathsf{ks}}, c_1 + c_1^{\mathsf{ks}})$ and denote the ciphertext term we want to remove by c_2. This also covers rotations as one only has to consider the term we want to remove as c_1 and an output of $(c_0 + c_1^{\mathsf{ks}}, c_1^{\mathsf{ks}})$. More specifically, we again make use of the RLWE hardness assumption to hide s^2 using s. Decryption with s "unboxes" s^2 and applies it to the ciphertext term c_2. In the following, we provide the general algorithms for key switching:

KeySwitchGen(s, s^2)

Receive secret key s^2 and secret key target s.
Sample $a \leftarrow \mathcal{U}_{q_L}$, $e \leftarrow \chi_e$ and output key switching key

$$\mathsf{ks} = (\mathsf{ks}_0, \mathsf{ks}_1) \equiv (-a \cdot s + te + s^2, a) \pmod{q_L}.$$

KeySwitch$(\mathsf{ks}, \mathfrak{c})$

Receive extended ciphertext $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and key switching key ks.
Switch key for $c_0 + c_1 \cdot s + c_2 \cdot s^2$ with

$$\mathbf{c}' \equiv (c_0 + c_2 \cdot \mathsf{ks}_0, c_1 + c_2 \cdot \mathsf{ks}_1) \pmod{q_\ell}$$

and output $(\mathbf{c}', \ell, \nu_{\mathsf{ks}})$.

Since q_ℓ divides q_L, $[\mathsf{ks}]_{q_\ell}$ is a valid key switching key with respect to q_ℓ and thus

$$\begin{aligned} c_0' + c_1' \cdot s &\equiv c_0 + c_2 \cdot \mathsf{ks}_0 + (c_1 + c_2 \cdot \mathsf{ks}_1) \cdot s && \pmod{q_\ell} \\ &\equiv c_0 - c_2 \cdot a \cdot s + c_2 \cdot te + c_2 \cdot s^2 + c_1 \cdot s + c_2 \cdot a \cdot s && \pmod{q_\ell} \\ &\equiv c_0 + c_1 \cdot s + c_2 \cdot s^2 + tc_2 \cdot e && \pmod{q_\ell}. \end{aligned}$$

Thus, the error after the key switching algorithm is bounded by

$$||\nu_{\mathsf{ks}}||^{can} = ||[c_0' + c_1' \cdot s]_{q_\ell}||^{can} \leq ||\nu||^{can} + tc_2 \cdot e.$$

Unfortunately, the error after the key switching algorithm grows too much with the term $tc_2 \cdot e$ and thus several variants exist to reduce its growth. In this work, we consider the three main variants: the Brakerski Vaikuntanathan (BV) variant, the Gentry Halevi Smart (GHS) variant, and the Hybrid variant.

BV The BV variant [4] decomposes c_2 with respect to a base ω to reduce the error growth. For polynomials α and β and $l = \lfloor \log_\omega q_\ell \rceil + 1$, we define

$$\mathcal{D}_\omega(\alpha) = ([\alpha]_\omega, [\lfloor \alpha/\omega \rfloor]_\omega, \ldots, [\lfloor \alpha/\omega^{l-1} \rfloor]_\omega)$$

$$\mathcal{P}_\omega(\beta) = ([\beta]_{q_\ell}, [\beta\omega]_{q_\ell}, \ldots, [\beta\omega^{l-1}]_{q_\ell})$$

and it follows that, for any $\alpha, \beta \in \mathcal{R}_{q_\ell}$, we have $\langle \mathcal{D}_\omega(\alpha), \mathcal{P}_\omega(\beta) \rangle \equiv \alpha \cdot \beta \bmod q_\ell$ [16].

KeySwitchGen$^{\mathsf{BV}}(s, s^2)$

Receive secret key s' and secret key target s.
Sample $\mathbf{a} \leftarrow \mathcal{U}_{q_L}^l$, $\mathbf{e} \leftarrow \chi_e^l$ and output key switching key

$$\mathsf{ks}^{\mathsf{BV}} = (\mathbf{ks}_0^{\mathsf{BV}}, \mathbf{ks}_1^{\mathsf{BV}}) = (-\mathbf{a} \cdot s + t\mathbf{e} + \mathcal{P}_\omega(s^2), \mathbf{a}) \pmod{q_L}.$$

KeySwitch$^{\mathsf{BV}}(\mathbf{ks}^{\mathsf{BV}}, \mathfrak{c})$

Receive extended ciphertext $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and key switching key $\mathsf{ks}^{\mathsf{BV}}$.
Switch key for $c_0 + c_1 \cdot s + c_2 \cdot s^2$ with

$$\mathbf{c}' = (c_0 + \langle \mathcal{D}_\omega(c_2), \mathbf{ks}_0^{\mathsf{BV}} \rangle, c_1 + \langle \mathcal{D}_\omega(c_2), \mathbf{ks}_1^{\mathsf{BV}} \rangle) \pmod{q_\ell}$$

and output $(\mathbf{c}', \ell, \nu_{\mathsf{ks}}^{\mathsf{BV}})$.

The error after the BV key switching is $c_0' + c_1' \cdot s \equiv c_0 + c_2 \cdot \mathsf{ks}_0^{\mathsf{BV}} + (c_1 + c_2 \cdot \mathsf{ks}_1^{\mathsf{BV}}) \cdot s \pmod{q_\ell}$, namely,

$$|| [c_0 + c_1 \cdot s + \langle \mathcal{D}_\omega(c_2), \mathcal{P}_\omega(s^2) \rangle + t \langle \mathcal{D}_\omega(c_2), \mathbf{e} \rangle]_{q_\ell} ||^{can},$$

that is,

$$||\nu_{\mathsf{ks}}^{\mathsf{BV}}||^{can} = ||[c_0' + c_1' \cdot s]_{q_\ell}||^{can} \leq ||\nu||^{can} + ||t \langle \mathcal{D}_\omega(c_2), \mathbf{e} \rangle||^{can}.$$

Since $t \langle \mathcal{D}_\omega(c_2), \mathbf{e} \rangle = t \sum_{i=0}^{l-1} [\lfloor c_2 / \omega^i \rfloor]_\omega \cdot e_i = t \sum_{i=0}^{l-1} \tilde{\omega}_i \cdot e_i$, we have

$$V_{t \cdot \langle \mathcal{D}_\omega(c_2), \mathbf{e} \rangle} = t^2 l \phi(m) V_{\tilde{\omega}_i} V_{e_i}.$$

We can assume that $\tilde{\omega}_i$ behaves like a uniform polynomial drawn from $\mathcal{U}_\omega$. So

$$||t \cdot \langle \mathcal{D}_\omega(c_2), \mathbf{e} \rangle||^{can} \leq D \sqrt{\phi(m) t^2 l \phi(m) \frac{\omega^2}{12} V_{e_i}} = D t \phi(m) \omega \sqrt{l \frac{V_e}{12}}.$$

Finally, we have $l = \sqrt{\lfloor \log_\omega(q_\ell) \rfloor + 1} \sim \sqrt{\log_\omega(q_\ell)}$ and can set

$$||\nu_{\mathsf{ks}}^{\mathsf{BV}}||^{can} \leq ||\nu||^{can} + \omega \sqrt{\log_\omega(q_\ell)} \mathsf{B}_{\mathsf{ks}},$$

where

$$\mathsf{B}_{\mathsf{ks}} = D t \phi(m) \sqrt{V_e / 12}. \tag{7}$$

GHS The GHS variant [11] switches to a bigger ciphertext modulus $Q_\ell = q_\ell P$ for a big prime P. Then, key switching takes places in $\mathcal{R}_{Q_\ell}$ and, by modulus switching back down to q_ℓ, the error is reduced again. As a tradeoff, we have to make sure that our RLWE instances are secure with respect to Q_ℓ.

KeySwitchGen$^{\mathsf{GHS}}(s, s^2)$

Receive secret key s^2 and secret key target s.
Sample $a \leftarrow \mathcal{U}_{Q_L}$, $e \leftarrow \chi_e$ and output key switching key

$$\mathsf{ks}^{\mathsf{GHS}} = (\mathsf{ks}_0^{\mathsf{GHS}}, \mathsf{ks}_1^{\mathsf{GHS}}) \equiv (-a \cdot s + te + P s^2, a) \pmod{Q_L}.$$

KeySwitch$^{\mathsf{GHS}}(\mathbf{ks}, \mathfrak{c})$

Receive extended ciphertext $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and key switching key $\mathsf{ks}^{\mathsf{GHS}}$.
For $c_0 + c_1 \cdot s + c_2 \cdot s^2$, switch key with

$$\mathbf{c}' \equiv (Pc_0 + c_2 \cdot \mathsf{ks}_0^{\mathsf{GHS}}, Pc_1 + c_2 \cdot \mathsf{ks}_1^{\mathsf{GHS}}) \bmod q_\ell.$$

Set $\boldsymbol{\delta} = t[-\mathbf{c}'t^{-1}]_P$, modulus switch back with

$$\mathbf{c}'' = \frac{1}{P}(\mathbf{c}' + \boldsymbol{\delta}) \pmod{q_\ell}$$

and output $(\mathbf{c}'', \ell, \nu_{\mathsf{ks}}^{\mathsf{GHS}})$.

Since we use modulus switching, showing correctness is similar in most aspects. For some $k \in \mathbb{Z}$, let $[c_0' + c_1' \cdot s]_{Q_\ell} = P[c_0 + c_1 \cdot s + c_2 \cdot s^2]_{q_\ell} + tc_2 \cdot e - kQ_\ell$.

$$\begin{aligned}[c_0'' + c_1'' \cdot s]_{q_\ell} &= [c_0 + c_1 \cdot s + c_2 \cdot s^2]_{q_\ell} + \frac{tc_2 \cdot e + \delta_0 + \delta_1 \cdot s}{P} \\ &\equiv m \bmod t.\end{aligned}$$

As in [6], we suppose that c_2 behaves like a uniform polynomial samples from $\mathcal{U}_{q_\ell}$ and, as before, $[-\mathbf{c}t^{-1}]_P$ behaves like a uniform polynomial samples from $\mathcal{U}_P$. Then,

$$\begin{aligned}||\nu_{\mathsf{ks}}^{\mathsf{GHS}}||^{can} &\leq |||[c_0 + c_1 \cdot s + c_2 \cdot s^2]_{q_\ell}||^{can} + \frac{||tc_2 \cdot e + \delta_0 + \delta_1 \cdot s||^{can}}{P} \\ &= ||\nu||^{can} + \frac{D\sqrt{\phi(m)(q_\ell^2 \frac{V_e}{12} + t^2P^2\frac{1}{12}(1 + \phi(m)V_s))}}{P} \\ &\leq ||\nu||^{can} + \frac{q_\ell}{P}\mathsf{B}_{\mathsf{ks}} + \mathsf{B}_{\mathsf{scale}},\end{aligned}$$

where B_{ks} and $\mathsf{B}_{\mathsf{scale}}$ are as in Equations (6) and (7), respectively. Decryption, and thus key switching, is correct as long as $||\nu||^{can} < \frac{q_\ell}{2c_m} - \frac{q_\ell}{P}\mathsf{B}_{\mathsf{ks}} + \mathsf{B}_{\mathsf{scale}}$.

Hybrid The Hybrid variant combines the BV and GHS variants [11]. In the following, we use the same notation from the variants as before.

KeySwitchGen$^{\mathsf{Hybrid}}(s, s^2)$

Receive secret key s^2 and secret key target s.
Sample $\mathbf{a} \leftarrow \mathcal{U}_{q_L}^l$, $\mathbf{e} \leftarrow \chi_e^l$ and output key switching key

$$\mathsf{ks}^{\mathsf{Hybrid}} = (\mathsf{ks}_0^{\mathsf{Hybrid}}, \mathsf{ks}_1^{\mathsf{Hybrid}}) \equiv (-\mathbf{a} \cdot s + t\mathbf{e} + P\,\mathcal{P}_\omega(s^2), \mathbf{a}) \pmod{Q_L}.$$

KeySwitch$^{\mathsf{Hybrid}}(\mathbf{ks}^{\mathsf{Hybrid}}, \mathfrak{c})$

Receive extended ciphertext $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and key switching key $\mathsf{ks}^{\mathsf{Hybrid}}$.
For $c_0 + c_1 \cdot s + c_2 \cdot s^2$, switch key with

$$\mathbf{c}' \equiv (Pc_0 + \langle \mathcal{D}_\omega(c_2), \mathsf{ks}_0^{\mathsf{Hybrid}} \rangle, Pc_1 + \langle \mathcal{D}_\omega(c_2), \mathsf{ks}_1^{\mathsf{Hybrid}} \rangle) \bmod Q_\ell.$$

Set $\boldsymbol{\delta} = t[-\mathbf{c}'t^{-1}]_P$, modulus switch back with

$$\mathbf{c}'' = \frac{1}{P}(\mathbf{c}' + \boldsymbol{\delta}) \pmod{q_\ell}$$

and output $(\mathbf{c}'', \ell, \nu_{\mathsf{ks}}^{\mathsf{Hybrid}})$.

Correctness follows by combining the proofs of each variant. The bounds also follow similarly, since before to scale down we have $\nu' = \nu P + \omega l \mathsf{B}_{\mathsf{ks}}$, where B_{ks} is as an Equation (7). Thus, the error after the modulus switching procedure is bounded by $\frac{q_\ell}{Q_\ell}||\nu'||^{can} + \mathsf{B}_{\mathsf{scale}}$, that is,

$$||\nu_{\mathsf{ks}}^{\mathsf{Hybrid}}||^{can} \leq ||\nu||^{can} + \frac{\omega\sqrt{\log_\omega(q_\ell)}}{P}\mathsf{B}_{\mathsf{ks}} + \mathsf{B}_{\mathsf{scale}},$$

where $\mathsf{B}_{\mathsf{scale}}$ is defined as Equation (6).

4 DCRT Representation

The Double Chinese Remainder Theorem (DCRT) changes the representation of the polynomials. This also influences the computations itself and thus slight adaptations to the bounds have to be made. In the following, we will briefly explain the DCRT representation and adjust the error bounds accordingly.

To represent polynomials in the DCRT representation, we need to apply two concepts based on the Chinese Remainder Theorem (CRT): the residue number system (RNS) and the Number Theoretic Transform (NTT). The RNS decomposes integers in $\mathbb{Z}_q$ into smaller integers $\mathbb{Z}_{q_i}$ for $q = \prod q_i$. For pairwise coprime q_i, we define a ring isomorphism $\mathbb{Z}_q \cong \mathbb{Z}_{q_1} \times \ldots \times \mathbb{Z}_{q_k}$ with

$$x \bmod q \mapsto (x \bmod q_1, \ldots, x \bmod q_k).$$

In the context of BGV, we decompose a polynomial in $\mathcal{R}_q$ into smaller polynomials in $\mathcal{R}_{q_1} \times \ldots \times \mathcal{R}_{q_2}$.

The Number Theoretic Transform (NTT) and its inverse, the INTT, transform a polynomial to a point-wise representation such that

$$\mathrm{INTT}(\mathrm{NTT}(a) \odot \mathrm{NTT}(b)) = a \cdot b$$

where $\odot$ denotes the point-wise multiplication of the transformed polynomials. This significantly reduces the cost of polynomial multiplication from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, the running time of the NTT. Mathematically, the NTT evalutes the polynomial in each of the $2n$ roots of unity. For a full definition, we refer the interested reader to [25].

4.1 DCRT Adaptations

For the DCRT representation, we have to make slight adaptations to our algorithms and, more importantly, to the error bounds. Key generation and encryption as well as addition, multiplication and constant multiplication only use trivial operations that can all be done in the DCRT domain. For decryption, we have to convert back to the coefficient domain with big integers right before removing the error, since the operation $m+te \bmod t$ does not work in the DCRT representation with respect to q_0. For modulus switching, we have to convert the DCRT polynomial for p_ℓ to the coefficient domain to calculate delta.

For key switching, we have to do slightly bigger adaptations for both the BV and GHS variants, thus also influencing the Hybrid variant. For the BV variant, we define $\mathcal{D}$ and $\mathcal{P}$ not with respect to some digit decomposition ω, but rather use the already existing RNS decomposition. For the proof, we only use the fact that $\langle \mathcal{D}(\alpha), \mathcal{P}(\beta) \rangle \equiv \alpha \cdot \beta \bmod q_\ell$ and thus we can use

$$\mathcal{D}(\alpha) = \left(\left[\alpha \left(\frac{q_\ell}{p_0} \right)^{-1} \right]_{p_0}, \ldots, \left[\alpha \left(\frac{q_\ell}{p_\ell} \right)^{-1} \right]_{p_\ell} \right) \text{ and } \mathcal{P}(\beta) = \left(\left[\beta \frac{q_\ell}{p_0} \right]_{q_\ell}, \ldots, \left[\beta \frac{q_\ell}{p_\ell} \right]_{q_\ell} \right).$$

Using the same approach as before, the error after the BV key switching is bounded by $||\nu||^{can} + ||t\langle \mathcal{D}(c_2), \mathbf{e} \rangle||^{can}$. Since $t\langle \mathcal{D}(c_2), \mathbf{e} \rangle = t \sum_{i=0}^{\ell} [c_2]_{p_i} \cdot e_i$ and c_2 behaves like a uniform polynomial samples from $\mathcal{U}_{p_\ell}$, we have

$$||\nu_{\mathsf{ks}}^{\mathsf{BV-DCRT}}||^{can} \leq ||\nu||^{can} + \mathsf{B}_{\mathsf{ks}} p_\ell \sqrt{\ell + 1}.$$

For the GHS variant, we also apply the RNS to P such that $P = P_1 \cdot P_2 \cdot \ldots \cdot P_k$. Then we extend to $Q_\ell = P q_\ell$ by calculating the following for each P_i:

$$\sum_{j=0}^{\ell} \left[c_2 \left(\frac{q_\ell}{p_j} \right)^{-1} \right]_{p_j} \frac{q_\ell}{p_j} \pmod{P_i}.$$

Note that this actually results in $[c_2]_{p_\ell} + \kappa p_\ell$ in the RNS base P and some polynomial κ. We then define $\mathbf{c}'$ as in Section 3.4 using the above extension resulting in $c_0' + c_1' \cdot s$ is equivalent $\bmod Q_\ell$ to

$$P[c_0 + c_1 \cdot s + (c_2 + \kappa p_\ell) \cdot s^2]_{p_\ell} + t([c_2]_{p_\ell} + \kappa p_\ell) \cdot e \equiv P[c_0 + c_1 \cdot s + c_2 \cdot s^2]_{p_\ell} + t([c_2]_{p_\ell} + \kappa p_\ell) \cdot e.$$

Note that, to perform the modulus switching, we have to consider $\boldsymbol{\delta}$ in base $\{p_0, \ldots, p_\ell\}$. So we have to convert back $\mathbf{c}'$ from $\{P_1, \ldots, P_k\}$ to $\{p_1, \ldots, p_\ell\}$, obtaining $\tilde{\boldsymbol{\delta}} = t([-\mathbf{c}' t^{-1}]_P + P\tilde{\kappa})$. Since $\mathbf{c}'' = \frac{1}{P}(\mathbf{c}' + \tilde{\boldsymbol{\delta}}) \pmod{q_\ell}$, we have that

$$[c_0'' + c_1'' \cdot s]_{p_\ell} = [c_0 + c_1 \cdot s + c_2 \cdot s^2]_{p_\ell} + \frac{t([c_2]_{p_\ell} + \kappa p_\ell) \cdot e + \tilde{\delta}_0 + \tilde{\delta}_1 \cdot s}{P}.$$

We again apply the same approach as before, and we obtain

$$||\nu_{\mathsf{ks}}^{\mathsf{GHS}}||^{can} \leq ||\nu||^{can} + \frac{t}{P} ||([c_2]_{p_\ell} + \kappa p_\ell) \cdot e||^{can} + ||\tilde{\delta}_0 + \tilde{\delta}_1 \cdot s||^{can}.$$

Since

$$[c_2]_{p_\ell} + \kappa p_\ell = \sum_{j=0}^{\ell} \left[c_2 \left(\frac{q_\ell}{p_j} \right)^{-1} \right]_{p_j} \frac{q_\ell}{p_j} \pmod{P_i}$$

and

$$\tilde{\delta} = t \sum_{j=1}^{k} \left[\boldsymbol{\delta} (\frac{P}{P_j})^{-1} \right]_{P_j} \frac{P}{P_j} \pmod{q_i},$$

we have that q_ℓ/p_j and P/P_j are constants and $[c_2 \left(q_\ell/p_j\right)^{-1}]_{p_j}$ and $[\boldsymbol{\delta}(P/P_j)^{-1}]_{P_j}$ behave like a uniform polynomial samples from $\mathcal{U}_{p_j}$ and $\mathcal{U}_{P_j}$, respectively. Thus,

$$\begin{aligned} V_{([c_2]_{p_\ell} + \kappa p_\ell) \cdot e} =& \phi(m)(\ell+1) \left(\frac{q_\ell}{p_j} \right)^2 V_{p_j} V_e = \phi(m)(\ell+1) q_\ell^2 \frac{V_e}{12}. \\ V_{\tilde{\delta}_0 + \tilde{\delta}_1 \cdot s} =& kt^2 \left(\frac{P}{P_j} \right)^2 V_{P_j} (1 + \phi(m) V_s) = kt^2 \frac{P^2}{12} (1 + \phi(m) V_s). \end{aligned}$$

So we have

$$||\nu_{\mathsf{ks}}^{\mathsf{GHS-RNS}}||^{can} \leq ||\nu||^{can} + \frac{q_\ell \sqrt{\ell+1}}{P} \mathsf{B}_{\mathsf{ks}} + \sqrt{k} \mathsf{B}_{\mathsf{scale}}.$$

For Hybrid, by combining both techniques, the resulting bound is

$$||\nu_{\mathsf{ks}}^{\mathsf{Hybrid-RNS}}||^{can} \leq ||\nu||^{can} + \frac{p_\ell \sqrt{\ell+1}}{P} \mathsf{B}_{\mathsf{ks}} + \sqrt{k} \mathsf{B}_{\mathsf{scale}}.$$

5 Studying the error growth and modulus size

In this section, we study the modulus size in a similar manner as in previous works [6, 7, 11]. For our model, we provide multiple options: an optional constant multiplication, η summands and τ rotations seperated by multiplications as shown in Figure 1.

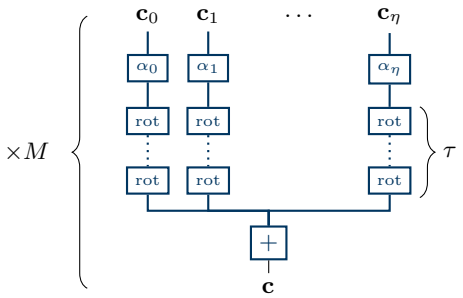

Fig. 1. Our analysis model depicted as circuit.

We also analyze the modulus sizes for a base model without rotations, providing better bounds for use cases without rotations. The "starting noise" of a ciphertext is denoted by B, in our case the noise after an encryption and modulus switching with the top modulus. Considering the multiplication of two ciphertexts, $\mathbf{c}_1 \cdot \mathbf{c}_2$, both with starting noise and subsequent operations according to our model, the noise grows to

$$(\eta||\text{CONST}||^{can}B + \eta\tau v_{ks})^2 = \varsigma^2 B^2 + \eta^2\tau^2 v_{ks}^2 + 2\eta\tau\varsigma v_{ks} B,$$

The added noise of the constant multiplication is $\varsigma = \eta||\text{CONST}||^{can} = \eta\mathsf{B}_{\mathsf{const}}$ with $\mathsf{B}_{\mathsf{const}} = Dt\sqrt{\frac{\phi(m)}{12}}$ (as in Section 3.2). The rotations itself do not influence the noise directly, the key switching back to the original key however adds key switching noise v_{ks}. Specifically, depending on the key switching method, we have

$$v_{ks} = \begin{cases} \omega\sqrt{\log_\omega(q_\ell)}\mathsf{B}_{\mathsf{ks}} & \text{(BV)} \\ q_\ell\mathsf{B}_{\mathsf{ks}}/P + \mathsf{B}_{\mathsf{scale}} & \text{(GHS)} \\ \omega\sqrt{\log_\omega(q_\ell)}\mathsf{B}_{\mathsf{ks}}/P + \mathsf{B}_{\mathsf{scale}} & \text{(Hybrid)} \end{cases} \tag{8}$$

Note that in the base model without rotations, the noise magnitude grows from B to ξB^2, where if we do

$$\begin{array}{ll} \star\ 1 \text{ multiplication, then } \eta \text{ additions} & \implies \xi = \eta\mathsf{B}_{\mathsf{const}}^2 \\ \star\ \eta \text{ additions, then 1 multiplication} & \implies \xi = \eta^2\mathsf{B}_{\mathsf{const}}^2 \\ \star\ \eta_1 \text{ additions for 1 } \textit{slot} \text{ of ciphertexts and} & \\ \quad \eta_2 \text{ for the second one, then} & \implies \xi = \eta_2\eta_1\mathsf{B}_{\mathsf{const}}^2. \\ \quad 1 \text{ multiplication between these two results} & \end{array} \tag{9}$$

Note that when we do not have any constant multiplication we set $\mathsf{B}_{\mathsf{const}} = 1$.

5.1 The Top Modulus

The top modulus is the first modulus in the prime chain, that is p_{L-1}. Before do any operation, we reduce the error $\mathsf{B}_{\mathsf{clean}}$ down to B, as in [11]. We use the modulus switching reducing the error and go down to next level:

$$\frac{\mathsf{B}_{\mathsf{clean}}}{p_{L-1}} + \mathsf{B}_{\mathsf{scale}} < B \iff p_{L-1} > \frac{\mathsf{B}_{\mathsf{clean}}}{B - \mathsf{B}_{\mathsf{scale}}} \tag{10}$$

5.2 The Bottom Modulus

The bottom modulus is the last modulus in the prime chain, that is p_0. At level zero, no more operations are performed. To ensure a correct decryption, we require that the noise is smaller than $q_0/2$ and thus $c_m B < q_0/2$. Based on Equation (18), it follows that

$$p_0 = q_0 > 4c_m\mathsf{B}_{\mathsf{scale}}. \tag{11}$$

5.3 Middle Moduli (when $L \geq 3$)

As described before, after a multiplication and additional operations, the noise magnitude grows from B to $(\varsigma B) + \eta\tau v_{ks})^2$. To reduce the noise magnitude down to B, we have to perform modulus switching. Additionally, we assume a key switching before the multiplication if the user prefers to delay the conversion of $c_2 \cdot s^2$.

For GHS and Hybrid key switching, we can actually merge the key switching with the modulus switching and directly switch down to a smaller modulus, that is from Q_ℓ to $q_{\ell-1}$ decreasing the noise by $q_{\ell-1}/Q_\ell = 1/(Pp_\ell)$. As before, the noise decreases due to the modulus switching, but we have to add the key switching noise **ks** depending on the method and the modulus switching noise $\mathsf{B}_{\mathsf{scale}}$. By merging the two operations, we get slightly improved bounds overall.

BV key-switching : Since we want to reduce the noise size after the key switching and the modulus switching back to B, we have to set

$$\frac{(\varsigma B + \eta\tau\omega\sqrt{\lfloor \log_\omega(q_\ell)\rfloor}\mathsf{B}_{\mathsf{ks}})^2}{p_\ell} + \frac{\omega\mathsf{B}_{\mathsf{ks}}\sqrt{\lfloor \log_\omega(q_\ell)\rfloor}}{p_\ell} + \mathsf{B}_{\mathsf{scale}} < B. \tag{12}$$

Note that $\sqrt{\lfloor \log_\omega(q_\ell)\rfloor} \sim \sqrt{(\ell+1)\log_\omega(p_\ell)}$ and, for p_ℓ enough big, we have that $\omega\mathsf{B}_{\mathsf{ks}}\sqrt{\lfloor \log_\omega(q_\ell)\rfloor}/p_\ell \sim 0$. Then,

$$\frac{\xi B^2}{p_\ell} + \Big(\frac{2\eta\tau\varsigma\omega\sqrt{(\ell+1)\log_\omega(p_\ell)}\mathsf{B}_{\mathsf{ks}}}{p_\ell} - 1\Big)B + \frac{(\eta\tau\omega\sqrt{(\ell+1)\log_\omega(p_\ell)}\mathsf{B}_{\mathsf{ks}})^2}{p_\ell} + \mathsf{B}_{\mathsf{scale}} < 0,$$

As Equation (12) must have a positive discriminant, we have

$$1 - \frac{4\eta\tau\varsigma\omega\sqrt{(\ell+1)\log_\omega(p_\ell)}\mathsf{B}_{\mathsf{ks}}}{p_\ell} - \frac{\xi\mathsf{B}_{\mathsf{scale}}}{p_\ell} \geq 0.$$

Moreover, the p_ℓ's have roughly the same size, thus we can set

$$p_1 \sim \ldots \sim p_{L-2} \sim 4\eta\varsigma(\tau\omega\sqrt{L-1}\log_\omega(\mathsf{B}_{\mathsf{ks}})\mathsf{B}_{\mathsf{ks}} + \mathsf{B}_{\mathsf{const}})\mathsf{B}_{\mathsf{scale}} \tag{13}$$

So we have $2\eta\tau\varsigma\omega\sqrt{(\ell+1)\log_\omega(p_\ell)}\mathsf{B}_{\mathsf{ks}}/p_\ell \sim 0$ and the discriminant ~ 0. Thus,

$$B \sim \frac{1}{2\xi/p_\ell} \sim 2\Big(\frac{\tau\omega\sqrt{L-1}\log_\omega(\mathsf{B}_{\mathsf{ks}})\mathsf{B}_{\mathsf{ks}}}{\mathsf{B}_{\mathsf{const}}} + 1\Big)\mathsf{B}_{\mathsf{scale}} \tag{14}$$

Previous work has shown, that $3 \leq \omega \leq 5$ is a good choice for ω [16].

If we do not have any rotation, that is $\tau = 0$, Equation (12) becomes

$$\frac{\xi B^2}{p_\ell} + \frac{\omega\mathsf{B}_{\mathsf{ks}}\sqrt{\lfloor \log_\omega(q_\ell)\rfloor}}{p_\ell} + \mathsf{B}_{\mathsf{scale}} < B.$$

As before, we must have a positive discriminant and thus

$$1 - \frac{4\xi}{p_\ell}\Big(\frac{\omega\mathsf{B}_{\mathsf{ks}}\sqrt{(\ell+1)\log_\omega(p_\ell)}}{p_\ell} + \mathsf{B}_{\mathsf{scale}}\Big) \sim 1 - \frac{4\xi}{p_\ell}\mathsf{B}_{\mathsf{scale}} \geq 0.$$

We set $p_1 \sim \ldots \sim p_{L-2} \sim 4\xi\mathsf{B}_{\mathsf{scale}}$ and $B \sim \frac{1}{2\xi/p_\ell} \sim 2\mathsf{B}_{\mathsf{scale}}$.

GHS key-switching : As before, we want to reduce the noise size from $(\varsigma B + \eta\tau v_{ks})^2$ back to B using key and modulus switching, where v_{ks} is as in (8). Let $\xi = \varsigma^2$, so we have

$$\frac{P \cdot (\varsigma B + \eta\tau v_{ks})^2 + \mathsf{B_{ks}} \cdot q_\ell}{P p_\ell} + \mathsf{B_{scale}} < B$$

and thus

$$\frac{\xi B^2}{p_\ell} + \frac{2\eta\tau\varsigma}{p_\ell}\Big(\frac{q_\ell}{P}\mathsf{B_{ks}} + \mathsf{B_{scale}}\Big)B + \frac{\eta^2\tau^2}{p_\ell}\Big(\frac{q_\ell}{P}\mathsf{B_{ks}} + \mathsf{B_{scale}}\Big)^2 + \frac{\mathsf{B_{ks}} q_{\ell-1}}{P} + \mathsf{B_{scale}} < B. \tag{15}$$

To solve this inequality for B, we follow the idea of Gentry *et al.* [11]. Let $R_\ell = \frac{\eta^2\tau^2}{p_\ell}(\frac{q_\ell}{P}\mathsf{B_{ks}} + \mathsf{B_{scale}})^2 + \frac{\mathsf{B_{ks}} q_\ell}{P p_\ell} + \mathsf{B_{scale}}$. Since R_ℓ increases with larger ℓ's, we have to satisfy this inequality for the largest modulus $\ell = L-2$. Moreover, $R_{L-2} > \mathsf{B_{scale}}$. Since we want that this term is as close to $\mathsf{B_{scale}}$ as possible, we have to set P large enough. Namely,

$$P > k\mathsf{B_{ks}} q_{L-2}/\mathsf{B_{scale}} \tag{16}$$

and so Equation (15) becomes $\frac{\xi B^2}{p_\ell} + (\frac{2\eta\tau\varsigma}{p_\ell}\mathsf{B_{scale}} - 1)B + \frac{\eta^2\tau^2}{p_\ell}\mathsf{B}_{\mathsf{scale}}^2 + \mathsf{B_{scale}} < 0$. Thus, to satisfy this equation, we again must have a positive discriminant and therefore

$$\Big(\frac{2\eta\tau\varsigma}{p_\ell}\mathsf{B_{scale}} - 1\Big)^2 - 4\frac{\xi}{p_\ell}\Big(\frac{\eta^2\tau^2}{p_\ell}\mathsf{B}_{\mathsf{scale}}^2 + \mathsf{B_{scale}}\Big) \geq 0 \iff 1 - \frac{4\eta\varsigma(\tau + \mathsf{B_{const}})}{p_\ell}\mathsf{B_{scale}} \geq 0$$

We then have

$$p_1 \sim \ldots \sim p_{L-2} \sim 4\eta\varsigma(\tau + \mathsf{B_{const}})\mathsf{B_{scale}} \tag{17}$$

and $R_{L-2} \sim \mathsf{B_{scale}}$. Finally, if we set p_ℓ as in (17) and receive a discriminate equal to zero, we can find B with

$$B \sim \Big(\tau/\mathsf{B_{const}} + 2\Big)\mathsf{B_{scale}}. \tag{18}$$

If we do not have any rotation, that is $\tau = 0$, Equation (15) becomes

$$\frac{\xi B^2}{p_\ell} + \frac{\mathsf{B_{ks}} q_{\ell-1}}{P} + \mathsf{B_{scale}} < B.$$

Equations (17) and (18) are the same, instead P can be decrease to $k\mathsf{B_{ks}} q_{L-3}/\mathsf{B_{scale}}$. Indeed, R_ℓ changes to $\frac{\mathsf{B_{ks}} q_\ell}{P p_\ell} + \mathsf{B_{scale}} = \mathsf{B_{ks}} q_{\ell-1}/P + \mathsf{B_{scale}}$ (see also [11]).

Hybrid key switching : The noise after the key switching and the modulus switching is at most

$$\frac{\big(\varsigma B + \eta\tau(\frac{\omega\sqrt{\log_\omega(q_\ell)}\mathsf{B_{ks}}}{P} + \mathsf{B_{scale}})\big)^2}{p_\ell} + \frac{\omega\mathsf{B_{ks}}\sqrt{\lfloor\log_\omega(q_\ell)\rfloor}}{P p_\ell} + \mathsf{B_{scale}} < B.$$

Following the same argument as before, we obtain the same equality for p_i and B and receive

$$P \geq k\omega \mathsf{B}_{\mathsf{ks}} \sqrt{\log_\omega(q_{L-2})} / \mathsf{B}_{\mathsf{scale}}.$$

In this case, if we do not have any rotation or any constant multiplication, the equations for P, p_ℓ and B are the same (setting $\tau = 0$ and $\mathsf{B}_{\mathsf{const}} = 1$, respectively).

5.4 The Largest Modulus

The ciphertext modulus is given by $q = p_0 p_{L-1} \prod_{\ell=1}^{L-2} p_\ell$. We summarize the result of each component in Table 1. The biggest modulus is $Q = Pq_{L-1}$. From Table 1, one can also see that the hybrid key switching provide smaller P.

Case	p_0	p_ℓ	p_{L-1}	P	
$\tau \neq 0$	$4\mathsf{B}_{\mathsf{scale}}$	$4\eta\varsigma(\tau\omega\sqrt{L-1}\log_\omega(\mathsf{B}_{\mathsf{ks}})\mathsf{B}_{\mathsf{ks}} + \mathsf{B}_{\mathsf{const}})\mathsf{B}_{\mathsf{scale}}$	$\frac{\mathsf{B}_{\mathsf{clean}}}{\left(\frac{2\tau\omega\sqrt{L-1}\log_\omega(\mathsf{B}_{\mathsf{ks}})\mathsf{B}_{\mathsf{ks}}}{\mathsf{B}_{\mathsf{const}}}+1\right)\mathsf{B}_{\mathsf{scale}}}$	–	BV
				$\frac{k\mathsf{B}_{\mathsf{ks}} q_{L-2}}{\mathsf{B}_{\mathsf{scale}}}$	GHS
		$4\eta\varsigma(\tau + \mathsf{B}_{\mathsf{const}})\mathsf{B}_{\mathsf{scale}}$	$\frac{\mathsf{B}_{\mathsf{clean}}}{\left(\frac{\tau}{\mathsf{B}_{\mathsf{const}}}+1\right)\mathsf{B}_{\mathsf{scale}}}$	$\frac{k\omega\mathsf{B}_{\mathsf{ks}}\sqrt{\log_\omega(q_{L-2})}}{\mathsf{B}_{\mathsf{scale}}}$	Hybrid
$\tau = 0$		$4\xi\mathsf{B}_{\mathsf{scale}}$	$\frac{\mathsf{B}_{\mathsf{clean}}}{\mathsf{B}_{\mathsf{scale}}}$	–	BV
				$\frac{k\mathsf{B}_{\mathsf{ks}} q_{L-3}}{\mathsf{B}_{\mathsf{scale}}}$	GHS
				$\frac{k\omega\mathsf{B}_{\mathsf{ks}}\sqrt{\log_\omega(q_{L-2})}}{\mathsf{B}_{\mathsf{scale}}}$	Hybrid

Table 1. Modulus size for $d = 2^\kappa$, where ξ is as in Equation (9).

6 Security Analysis for BGV

In this section, to enable a more flexible parameter selection, we propose an empirically derived formula linking the security level λ with the dimension n for a given ciphertext modulus size $\log q$.

The LWE problem consists of finding the secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, given $\mathbf{b} \in \mathbb{Z}_q^m$ and $A \in (\mathbb{Z}_q)^{m \times n}$ such that $A\mathbf{s} + \mathbf{e} = \mathbf{b} \bmod q$, where $\mathbf{e} \in \mathbb{Z}_q^m$ is sampled from the error distribution χ_e. The security of LWE-based schemes depends on the intractability of this problem and attacks on these schemes are based on finding efficient algorithms to solve them [18]. In [3], the authors presented three different methodologies to solve the LWE problem and the central part of two of them is based on lattice reduction. Namely, starting from a bad (i.e. long) lattice basis, find a better (i.e. reduced and more orthogonal) basis.

The most well-known lattice reduction algorithm used in practice is BKZ (block Korkin-Zolotarev reduction) due to Schnorr and Euchner [24]. In these algorithms, the time complexity and the outcome quality (i.e. the orthogonality of the reduced basis) is characterised by the Hermite factor [9]. Specifically,

the run time of the BKZ algorithm is higher when the root-Hermite factor δ_0 is smaller [24]. This result is also supported by a realistic estimation provided in [3], where the authors show that the log of the time complexity to get a root-Hermite factor δ_0 with BKZ is

$$\log(t_{BKZ})(\delta_0) = \Omega\left(-\frac{\log(\log \delta_0)}{\log \delta_0}\right) \tag{19}$$

if calling the SVP oracle costs $2^{O(\beta)}$, where β is the the block-size of BKZ algorithm.

Since we need a formula linking λ with n and q, we have to find a relation between δ_0 with n and q. To do that, let us consider a full rank lattice $\mathcal{L}$. By Equation (4), we know that the shortest vector of $\mathcal{L}$ has norm $||\mathbf{b}_1|| = \delta_0^k q^{n/k}$. To perform lattice reduction on $\mathcal{L}$, the LWE attacker has to choose the number of semples M, namely the subdimension, such that $||\mathbf{b}_1|| = \delta_0^M q^{n/M}$ is minimized. Micciancio and Regev [20] showed that this minimum is obtained when $M = \sqrt{n \log q / \log \delta_0}$. Following the same approach of [11], we can suppose that we should reduce the basis enough so that $||\mathbf{b}_1|| = q$. This implies that $\log q = \log(\delta_0^M q^{n/M}) = 2\sqrt{n \log q \log \delta_0}$, that is,

$$n = \log q/(4 \log \delta_0). \tag{20}$$

Substituting (20) in (19), we have a bound linking λ, n and q. Then, we improve this bound finding a function that *follows* the data points generated with the lattice estimator [3]. Finally, we model the resulting formula with coupled optimization finding the final constants and we obtain the following

$$\lambda \approx -\log\left(\frac{\mathrm{A}\log q}{n}\right)\frac{\mathrm{B}n}{\log q} + \mathrm{C}\sqrt{\frac{\log q}{n}}\log\left(\frac{n}{\log q}\right), \tag{21}$$

where A = 0.07, B = 0.34 and C = 18.53 for $\chi_s = \mathcal{U}_3$ (Figure 2). This formula (provide λ, receive n) opens up the possibility to generate parameters for a given security level.

7 A Parameter Generator for BGV

The parameter generator for BGV aims to provide an accessible way to our theoretical work. Most importantly, developers can use the generator and receive a simple code example as well as a simple benchmarking setup to easily compare different parameter scenarios on their local setups. The generator itself is written in Python and available on GitHub [5]. It consists of four modules: the BGV module, a code generation module, a configuration module and an interactive module.

[5] `https://github.com/Crypto-TII/fhegen`

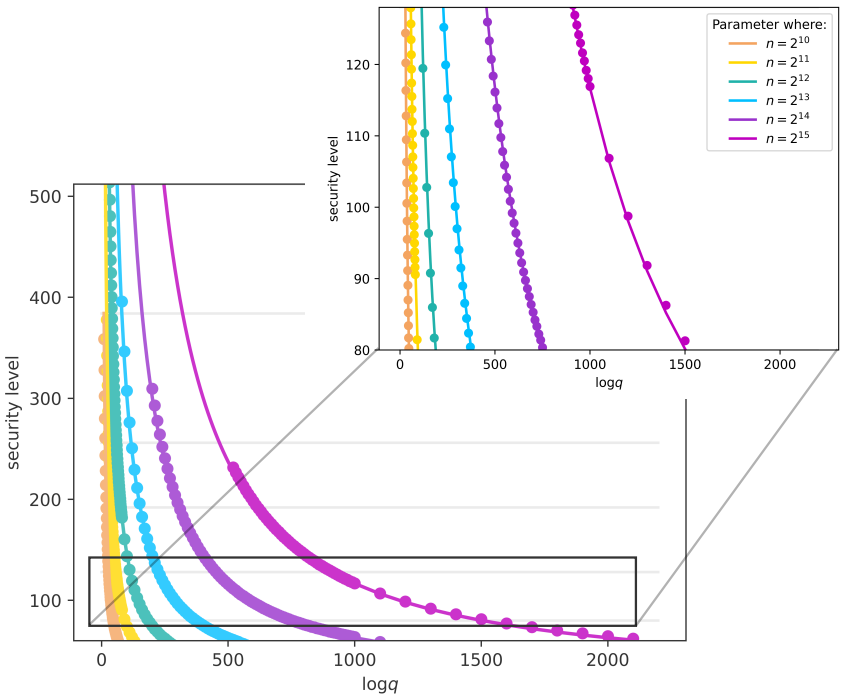

Fig. 2. The security formula with data points of the Lattice Estimator

The BGV module is the heart of the generator. It provides a function to calculate all necessary bound constants as well as generation functions for the ciphertext modulus q, a power-of-two cyclotomic order m and the key switching modulus P. Additionally, it handles the logic of the interactive menu and implements compatibility checks for both libraries, HElib and PALISADE.

The code generation module outputs three files if a user chooses one of the libraries: a Makefile, a `main.cpp` file with example and comparison code for a single plaintext and a `bench.cpp` file with simple benchmarks for the basic homomorphic operations. As we cannot cover all possible environments, users can adjust some of the environment-specific options such as the benchmark repetitions or the include paths for each library in the configuration module. The interactive module handles user dialogs as well as input parsing.

7.1 Interactive Mode

The interactive mode of the parameter generator prompts the user for a number of questions. We list required inputs in the first part, optional inputs in the second part of Table 2. For each input, the generator presents the user with a default value.

t or $\log t$	any integer ≥ 2
λ or m	any integer ≥ 40 or ≥ 4, respectively
M, η	any integer > 0
τ	any integer ≥ 0
Library	`'None'`, `'HElib'`, `'PALISADE'`
Full Batching	full batching with t, `'True'` or `'False'`
Secret Distribution	`'Ternary'`, `'Error'`
Key Switching	`'Hybrid'`, `'BV'`, `'GHS'`
ω	any integer ≥ 1

Table 2. Required and optional inputs to the parameter generator

Currently, the generator is only supporting powers of two, but we plan to extend support to general cyclotomic orders in a future version.

After providing all required information, the user receives the output in text form and, if a library was chosen, the generated code. The output for the ciphertext modulus contains the bound on the ciphertext modulus itself as well as the bounds for the bottom, middle and top modulus, respectively. An exemplary output is the following.

```
$ python fhegen/bgv.py
Welcome to the interactive parameter generator for BGV! :)

Do you want a specific plaintext modulus? [N/?]:
[...]
Do you want to continue to the advanced settings? [N/y]:

Generated your BGV configuration!
sec:   147.55
d:     16384
t:     65537
logq:  405 (32, 33, 4)
logP:  11
slots: 16384
Generating Makefile, main.cpp and bench.cpp for PALISADE.
```

Listing 1.3. Shortened example on usage and output of the parameter generator

7.2 Limitations

Although we try to generate working parameters for HElib and PALISADE, we cannot guarantee this due to the inner workings of each library. As both are rather sparse on detailed documentation, we tried to balance integrating constraints of both libraries with the desire to share the parameter generator so other people can use it. Together with the community, we want to continue improving the generator; one of these improvement ideas is to reduce the differences to both libraries more accurately.

With respect to HElib for example, we will need to integrate the same prime generation and automatic noise management in the parameter generator to make sure that the parameters match. Another example with PALISADE is the use of RNS modulus and key switching. Although we provide the theoretical bounds in this work, we did not yet integrate them into the generator. We nonetheless believe that the generator can and will aid users in parameter selection and is a good step towards a more usable and approachable BGV.

8 Conclusion

Finding an optimal set of parameters for a specific FHE scheme is a challenging task. For example, in the BGV scheme the complexity (i. e. depth) of the function to be homomorphically evaluated impacts the error growth. Higher depths require a larger ciphertext modulus q and the adoption of a larger modulus decreases the security level. The security level can be raised again by adopting a higher polynomial degree, but this impacts efficiency [18].

Costache *et al.* [7], improving a previous performance evaluation study [6], proposed theoretical bounds to estimate error growth, which can be used to heuristically find parameters. For this, they follow the same methodology as in [11] and improve the bound using Iliashenko's method [15]. However, the main focus of these papers is the comparison of different FHE schemes. Thus, we improve on their circuit model and provide a tool to generate these parameters.

There also exists prior work aimed at FHE parameterization for specific use-cases. Reagan *et al.* [22] a framework that automatically tunes FHE parameters for neural networks. CinguParam [14] focuses on generating Brakerski Fan Vercauteren (BFV) parameters and is also included in Cingulata [5], an FHE compiler. Recently, Gouert *et al.* [12] suggested brute-forcing $\log q$ to find a working parameter set. Most importantly, no use-case focused and easy-to-use tool for BGV parameter generation exists.

Acknowledgements We want to thank Anna Hambitzer for her helpful comments on coupled optimization.

References

1. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (CSUR) **51**(4), 1–35 (2018)
2. Albrecht, M.R., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption security standard. Tech. rep., `HomomorphicEncryption.org`, Toronto, Canada (November 2018)
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)

4. Brakerski, Z., Vaikuntanathan, V.: Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. pp. 505–524. Springer, Berlin, Heidelberg (2011)
5. Cingulata: `https://github.com/CEA-LIST/Cingulata`
6. Costache, A., Smart, N.P.: Which ring based somewhat homomorphic encryption scheme is best? In: Cryptographers' Track at the RSA Conference. pp. 325–340. Springer (2016)
7. Costache, A., Laine, K., Player, R.: Evaluating the effectiveness of heuristic worst-case noise analysis in FHE. In: European Symposium on Research in Computer Security. pp. 546–565. Springer (2020)
8. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Annual Cryptology Conference. pp. 643–662. Springer (2012)
9. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N. (ed.) Advances in Cryptology – EUROCRYPT 2008. pp. 31–51. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
10. Gentry, C.: A fully homomorphic encryption scheme, vol. 20. Stanford university Stanford (2009)
11. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic Evaluation of the AES Circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. pp. 850–867. Springer, Berlin, Heidelberg (2012)
12. Gouert, C., Khan, R., Tsoutsos, N.G.: Optimizing homomorphic encryption parameters for arbitrary applications. Cryptology ePrint Archive (2022)
13. Halevi, S., Shoup, V.: Algorithms in helib. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 554–571. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
14. Herbert, V.: Automatize parameter tuning in ring-learning-with-errors-based leveled homomorphic cryptosystem implementations. Cryptology ePrint Archive (2019)
15. Iliashenko, I.: Optimisations of fully homomorphic encryption (2019)
16. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields (2021)
17. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. pp. 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
18. Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F.H., Aaraj, N.: Survey on fully homomorphic encryption, theory and applications (2022)
19. Martins, P., Sousa, L., Mariano, A.: A survey on fully homomorphic encryption: An engineering perspective. ACM Computing Surveys (CSUR) **50**(6), 1–33 (2017)
20. Micciancio, D., Regev, O.: Lattice-based Cryptography. Springer, Berlin, Heidelberg (2009)
21. PALISADE: `https://palisade-crypto.org`
22. Reagen, B., Choi, W.S., Ko, Y., Lee, V.T., Lee, H.H.S., Wei, G.Y., Brooks, D.: Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 26–39. IEEE (2021)
23. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing. pp. 84–93 (2005)

24. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming **66**(1-3), 181–199 (1994)
25. Seiler, G.: Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography (January 2018), `https://eprint.iacr.org/2018/039`
26. Shai Halevi and Victor Shoup: HElib. `https://github.com/homenc/HElib`