Determination and exploration of practical parameters for the latest Somewhat Homomorphic Encryption (SHE) Schemes

Vincent Migliore, Guillaume Bonnoron, Caroline Fontaine

Determination and Exploration of Practical Parameters for the Latest *Somewhat* Homomorphic Encryption (SHE) Schemes.

Vincent Migliore[1], Guillaume Bonnoron[2], Caroline Fontaine[3]

[1] Université Bretagne Sud, Lab-STICC
[2] Chair of Naval Cyber Defense
[3] CNRS and Telecom Bretagne, Lab-STICC
vincent.migliore@univ-ubs.fr
guillaume.bonnoron@ecole-navale.fr
caroline.fontaine@imt-atlantique.fr

Abstract. Homomorphic encryption gets increasing attention lately, and for good reasons. A lot of the burdens from the initial proposals have been overcome and real applications become reachable. In this work, we propose a study of the current best solutions, providing a deep analysis of how to setup and size their parameters. Our overall aim is to provide easy-to-use guidelines for implementation purposes.

1 Introduction

Homomorphic Encryption (HE) is a recent promising tool in modern cryptography, that allows to carry out operations on encrypted data. Historically, some early cryptographic schemes presented partial homomorphic properties, for multiplication [12] or addition [22]. But it was only with the works from [21] and [13] that key ideas were introduced to support both operations simultaneously. These schemes have been followed by many others [4,9,6,5,11,14,7,16]. It is important to notice that nearly all these post-2009 schemes are built upon lattices, which makes a great difference when comparing with former partial HE schemes, both for performance (lattice-based homomorphic schemes lead to more practical constraints in terms of efficiency and encrypted data size) and for security considerations (lattice-based schemes security has been less studied, yet seems not affected by quantum cryptanalysis). In this paper, we will only focus on these post-2009 schemes built upon lattices, which enable both additions and multiplications over encrypted data. Among HE schemes, *Fully Homomorphic Encryption* (FHE) schemes allow the two types of elementary operations, without any restrictions, thus enabling to process virtually any algorithm over encrypted data. However the first FHE schemes presented too many drawbacks for a concrete use, as they had very high complexity and poor flexibility. So, to lighten the overhead of homomorphic capabilities, a more promising rationale has been investigated, the so-called *Somewhat Homomorphic Encryption* (SHE) schemes. These allow any

number of additions, but only a limited number of multiplications. By (upper-)bounding the number of homomorphic operations, SHE schemes considerably reduce the size of ciphertexts and associated costs. Among the many HE schemes that have been presented, the most promising ones are based on ideal lattices. Here, we focus on 2^{nd} and 3^{rd} generation schemes, which are the most efficient. For many years, the theoretical background of homomorphic encryption schemes has been evolving. Thus, it has remained a real challenge to draw practical parameters. Morevover, former publications usually present values for specific use-cases and do not address a wide range of applications. This issue stands in the way toward broader implementation and use of homomorphic encryption, therefore to address this, we concisely and precisely present here how the extraction of SHE parameters works. We make specific efforts to offer ready-to-use content to people from outside the cryptography community, providing pre-computed tables and simple formulas for a self-determination of parameters. We decided to focus our study on two of most promising schemes: FV[11] and SHIELD[16]. Usually, implementations target YASHE'[4] instead of FV. However confidence in this scheme has been recently damaged by the subfield/sublattice attack [1,17]. Making YASHE' immune to these attacks would lead to oversize its parameters, far too much for practical use. SEAL [18] moved to FV after these attacks. We selected SHIELD for comparison because it is a 3^{rd} generation scheme equivalent to FV. With larger costs for the first homomorphic multiplications, 3^{rd}-gen schemes have a much better asymptotic behavior. Also from this generation, F-NTRU [9] had to be discarded together with YASHE'. Another interesting candidate would have been BGV[6], used in HElib [15]. However BGV could not be fairly compared because of the modulus-switching operations that prevent efficient hardware implementations. Also we did not include schemes like [10,8] because they fall out of our discussion on somewhat homomorphic encryption.
The main contributions of the paper are: a concise presentation of the two schemes with harmonized notation; a review of parameters extraction for both of them, with several explorations to evaluate parameters for various applications; numerous tables of parameters under different constraints, in order to cover a wide range of use cases. It is organized as follows. Section 2 provides notation and the basic theoretical background. Section 3 presents FV and SHIELD with harmonized notation and provides a brief state of the art of current implementation techniques. Section 4 discusses the methodology for parameters extraction. Section 5 offers ready-to-use tables and compare these schemes according to different scenarios. Section 6 draws some conclusions.

2 Preliminaries

Let $\mathbb{Z}_q[X] = (\mathbb{Z}/q\mathbb{Z})[X]$ be the set of polynomials with integer coefficients modulo q. The m^{th} cyclotomic polynomial of degree n is noted $\Phi_m(X)$. We define $R_q = \mathbb{Z}_q[X]/\Phi_m(X)$ the ring of polynomials with integer coefficients modulo q, reduced by the cyclotomic polynomial $\Phi_m(X)$. A polynomial is represented with an uppercase and its coefficients with a lowercase. For polynomial A, a_i

represents its i^{th} coefficient. A vector of polynomials is noted in bold. For vector $\mathbf{A}$, $\mathbf{A}[i]$ is the i^{th} polynomial of the vector. For a set R and a polynomial A, $A \leftarrow U_R$ represents a uniformly sampled polynomial in R, $A \leftarrow B_R$ a uniformly sampled polynomial in R with binary coefficients and $A \leftarrow D_{R,\sigma}$ a polynomial of R with coefficients sampled from a discrete Gaussian distribution with width parameter σ, *i.e.* proportional to $\exp(-\pi x^2/\sigma^2)$. For coefficient a_i of polynomial A, $a_{i,(j..k)}$ corresponds to the binary string extraction of a_i between bits j and k. This notation is extended to polynomial A where $A_{(j..k)}$ is the sub-polynomial where the binary string extraction is applied to each coefficient. A modular reduction by an integer q is noted $[\cdot]_q$. For integer a, $\lfloor a \rfloor$, $\lceil a \rceil$ and $\lfloor a \rceil$ operators are respectively the floor, ceil and nearest rounding operations. This notation is extended to polynomials by applying the operation on each coefficient. For vectors $\mathbf{A}$ and $\mathbf{B}$, $\langle \mathbf{A}, \mathbf{B} \rangle$ represents $\sum \mathbf{A}[i]\mathbf{B}[i]$. To simplify notation, we use several variables: $l = \log_2 q$, $N = 2\ l$ and $l_{\omega,q} = \lceil \log_2 q / \log_2 \omega \rceil$, for some integer ω. In the following, all polynomial operations are considered performed in R_q.

We recall here the definition of the Ring-Learning With Errors problem [20].

Definition *Let R be a ring of degree n over $\mathbb{Z}$ (usually $R = \mathbb{Z}[x]/(f(x))$ for some cyclotomic polynomial $f(x)$). Let q be a positive integer, χ a probability distribution on R of width parameter σ and S a secret random element in R_q. We denote by $L_{S,\chi}$ the probability distribution on $R_q \times R_q$ obtained by choosing $A \in R_q$ uniformly at random, choosing $E \in R$ according to χ and considering it in R_q, and returning $(A, C) = (A, [A \cdot S + E]_q) \in R_q \times R_q$.*

Decision-Ring-LWE *is the problem of deciding whether given pairs (A, C) are sampled according to $L_{S,\chi}$ or the uniform distribution on $R_q \times R_q$.* Search-Ring-LWE *is the problem to recovering S from pairs (A, C) sampled from $L_{S,\chi}$.*

The hardness of Ring-LWE problem depends on the three variables n, σ and q. The reduction presented in the introductory paper stands when $\sigma > 2\sqrt{n}$.

3 Presentation of the schemes

3.1 FV

FV [11] is a transposition of the scale-invariant Brakerski scheme [5] to the Ring-LWE problem. Published at the same time as YASHE, it does not suffer from any security flaw and has been addressed as a very promising scheme in several recent publications. The public key is a pair $(AS + E, A)$ of a Ring-LWE instance, and the secret key is the polynomial S. After an homomorphic multiplication, the ciphertext is composed of 3 terms instead of 2. To recover its initial form, an additional step called relinearization is required, making use of a relinearization key. FV also introduces two additional parameters, namely t and ω. Integer $t \in \{1, \ldots, q\}$ corresponds to the upper bound of a message. When $t = 2$, messages are binary. ω is a parameter associated with the relinearization, and determines the size of the relinearization key and the complexity of the relinearization operation. It is usual to select ω as a 32-bit or 64-bit integer for computational aspects.

– $\text{FV.PowersOf}_{w,q}(A)$:
  $\mathbf{A} \in R_q^{l_{w,q}}$
  for $i = 0$ to $l_{w,q} - 1$
    $\mathbf{A}[i] = \left[Aw^i\right]_q$
  end for
  return $\mathbf{A}$

– $\text{FV.WordDecomp}_{w,q}(A)$:
  $\mathbf{A} \in R_q^{l_{w,q}}$
  for $i = 0$ to $l_{w,q} - 1$
    $l_0 = i \times \log_2 \omega$
    $l_1 = (i+1) \times \log_2 \omega - 1$
    $\mathbf{A}[i] = A_{(l_0..l_1)}$
  end for
  return $\mathbf{A}$

$\left\langle \text{FV.PowersOf}_{w,q}(A), \text{FV.WordDecomp}_{w,q}(B) \right\rangle = \left[A \times B\right]_q$.

– $\text{FV.GenKeys}(\lambda)$:
  $S \leftarrow D_{R_q,\sigma_{key}}$, $A \leftarrow U_{R_q}$, $E \leftarrow D_{R_q,\sigma_{err}}$
  $P_{key} = (-AS + E, A)$
  $S_{key} = S$
  return (P_{key}, S_{key})

– $\text{FV.GenRelinKeys}(P_{key}, S_{key})$:
  $\mathbf{A} \leftarrow U_{R_q}^{l_{w,q}}$, $\mathbf{E} \leftarrow D_{R_q,\sigma_{err}}^{l_{w,q}}$
  $\gamma = \left(\left[\text{FV.PowersOf}_{w,q}(S_{key}^2) - (\mathbf{A}S_{key} + \mathbf{E})\right]_q, \mathbf{A}\right)$
  return γ

– $\text{FV.Encrypt}(m, P_{key})$:
  $U \leftarrow D_{R_q,\sigma_{key}}^{l_{w,q}}$, $(E_1, E_2) \leftarrow D_{R_q,\sigma_{err}}^2$
  $C = \left(\left[\frac{q}{t}m + P_{key}[0]U + E_1\right]_q, \left[P_{key}[1]U + E_2\right]_q\right)$
  return C

– $\text{FV.Decrypt}(C, S_{key})$:
  $\widetilde{M} = \left[C[0] + C[1]S_{key}\right]_q$
  $m = \left\lfloor \frac{t}{q}\widetilde{M}[0] \right\rceil$
  return m

– $\text{FV.Add}(C_A, C_B)$:
  $C_+ = \left(\left[C_A[0] + C_B[0]\right]_q, \left[C_A[1] + C_B[1]\right]_q\right)$
  return C_+

– $\text{FV.Mult}(C_A, C_B, \gamma)$:
  $\widetilde{C}_0 = \left[\left\lfloor \frac{t}{q}C_A[0] \times C_B[0] \right\rceil\right]_q$
  $\widetilde{C}_1 = \left[\left\lfloor \frac{t}{q}\left(C_A[0] \times C_B[1] + C_A[1] \times C_B[0]\right) \right\rceil\right]_q$

$\widetilde{C}_2 = \left[\left\lfloor \frac{t}{q} C_A[1] \times C_B[1] \right\rceil\right]_q$
$C_\times = \text{FV.Relin}(\widetilde{C}_0, \widetilde{C}_1, \widetilde{C}_2, \gamma)$
return $C_\times$

– $\text{FV.Relin}(\widetilde{C}_0, \widetilde{C}_1, \widetilde{C}_2, \gamma)$:
$C_R = (C_{R,0}, C_{R,1})$
$C_{R,0} = \left[\widetilde{C}_0 + \left\langle \text{FV.WordDecomp}_{w,q}\ (\widetilde{C}_2), \gamma[0] \right\rangle\right]_q$
$C_{R,1} = \left[\widetilde{C}_1 + \left\langle \text{FV.WordDecomp}_{w,q}\ (\widetilde{C}_2), \gamma[1] \right\rangle\right]_q$
return C_R

3.2 SHIELD

SHIELD [16] is a transposition of the GSW scheme [14] to the Ring-LWE problem. It is a so called 3^{rd} generation HE schemes, and does not require any relinearization, but requires much more polynomials per ciphertext (namely $2 \times N = 4 \cdot \log_2 q$ for SHIELD, instead of 2 for FV). To counterbalance, the inner noise grows more slowly than in 2^{nd} generation HE schemes, reducing the size of the modulus q and the cyclotomic polynomial degree n. By carefully examining SHIELD, one can notice strong similarities with FV, especially for the key generation, the encryption and the decryption. Because no relinearization is required, the homomorphic multiplication is much more natural than in FV.

– $\text{SHIELD.BD}(\mathbf{A})$:
$(\mathbf{A} \in R_q^{N \times 2})$
$\mathbf{B} \in B_{R_q}^{N \times N}$
for $i = 0$ to $N - 1$
  for $j = 0$ to $\log_2 q - 1$
    $\mathbf{B}[i][j] = \mathbf{A}[i][0]_{(j)}$
    $\mathbf{B}[i][j + \log_2 q] = \mathbf{A}[i][1]_{(j)}$
  end for
end for
return $\mathbf{B}$

– $\text{SHIELD.BDI}(\mathbf{A})$:
$(\mathbf{A} \in B_{R_q}^{N \times N})$
$\mathbf{B} \in R_q^{N \times 2}$
for $i = 0$ to $N - 1$
  $\mathbf{B}[i][0] = \sum_{j=0}^{\log_2 q - 1} \mathbf{A}[i][j] 2^j$
  $\mathbf{B}[i][1] = \sum_{j=\log_2 q}^{N-1} \mathbf{A}[i][j] 2^j$

end for
return $\mathbf{B}$

- SHIELD.GenKeys(λ) :
  $T \leftarrow D_{R_q,\sigma_{key}}$, $A \leftarrow U_{R_q}$, $E \leftarrow D_{R_q,\sigma_{err}}$
  $B = A \cdot T + E$
  $\mathbf{P_{key}} = \begin{bmatrix} B & A \end{bmatrix}$
  $\mathbf{S_{key}} = \begin{bmatrix} 1 \\ -T \end{bmatrix}$
  return $(\mathbf{P_{key}}, \mathbf{S_{key}})$
- SHIELD.Encrypt($m, \mathbf{P_{key}}$) :
  $\mathbf{r_{N\times 1}} \leftarrow B_{R_q}^{N\times 1}$, $\mathbf{E_{N\times 2}} \leftarrow D_{R_q,\sigma_{err}}^{N\times 2}$
  $\mathbf{C} = \mathbf{C_{N\times 2}} = m \cdot \mathrm{BDI}(\mathbf{I_{N\times N}}) + \mathbf{r_{N\times 1}} \cdot \mathbf{P_{key}} + \mathbf{E_{N\times 2}}$
  return $\mathbf{C}$
- SHIELD.Decrypt($\mathbf{C}, \mathbf{S_{key}}$) :
  $\mathbf{M} = \mathbf{C} \cdot \mathbf{S_{key}} = m \cdot \mathrm{BDI}(\mathbf{I_{N\times N}}) \cdot \mathbf{S_{key}} + error$
  $m = \left\lfloor \frac{2}{q} \cdot \mathbf{M}[0][0] \right\rceil$
  return m
- SHIELD.Add($\mathbf{C_1}, \mathbf{C_2}$) :
  $\mathbf{C}_+ = \mathbf{C_1} + \mathbf{C_2}$
  return $\mathbf{C}_+$
- SHIELD.Mult($\mathbf{C_1}, \mathbf{C_2}$) :
  $\mathbf{C}_\times = \mathrm{BD}(\mathbf{C_1}) \cdot \mathbf{C_2}$
  return $\mathbf{C}_\times$

3.3 Batching

For each scheme above, the cleartext is a polynomial in R_q. For convenience, messages are commonly chosen to be integers. However, this integer representation turns out to be limited when considering interesting homomorphic operations. More evolved algorithms, *e.g.* calling comparison operators, require dealing with binary messages. This latter representation brings two important issues. First, to perform an integer addition or multiplication with the binary representation, one must reconstruct the binary circuit of the operators. Second, the size of ciphertexts is strongly impacted. To balance the ciphertext expansion issue, the batching technique is a good solution. Introduced in [24], the batching allows to "pack" several messages into one single ciphertext. To do so, the associated cyclotomic polynomial must be reducible modulo 2, and have only simple root factors. Then, a polynomial CRT is applied to pack the messages, with one message per factor.

3.4 Current implementation techniques and their impacts

Since the chosen polynomial multiplication algorithm impacts the parameters, we briefly introduce the Number Theoretic Transform (NTT) algorithm and its

NWC variant. To be efficient, NTT must be generated by a polynomial with irreducible factors of very small degree. This is why $x^n - 1$ and $x^n + 1$ are often chosen to be completely factorized with degree-1 factors. When performing a polynomial multiplication using the NTT algorithm, the output polynomial is reduced by the polynomial that generates the NTT, so it implies to double the size of the NTT w.r.t. the input polynomials. Also, $x^n + 1$ is a cyclotomic polynomial, and selecting this polynomial to generate the NTT provides a solution where the polynomial reduction is directly integrated into the computation. This special NTT is called *Negative Wrapped Convolution* (NWC) and requires a NTT of size n instead of $2n$ in the standard case. However, this cyclotomic property has an important issue. When factoring $x^n + 1$ modulo 2, the resulting polynomial is $(x+1)^n$, which has a unique factor, namely $(x+1)$. This is incompatible with the batching technique presented in Section 3.3. Thus, the NWC is optimized for performance but incompatible with batching techniques.

4 Parameters extraction

As described in Section 3.3, SHE proposes two types of evaluations: an operation on integer messages and binary messages. The following section focuses on the binary approach including also an exploration of the impact of the NWC NTT and the batching technique.

4.1 Noise management

4.1.1 Notation We briefly introduce additional notation for the noise extraction. For polynomials A and B, we define $\|A\|_\infty = \max_{0 \leq i < n} \mid a_i \mid$. When $A \leftarrow D_{R_q,\sigma_{key}}$ and $B \leftarrow D_{R_q,\sigma_{err}}$, we note $\|A\|_\infty = B_{key}$ and $\|B\|_\infty = B_{err}$. B_0 refers to the upper bound of the noise for a fresh ciphertext, B_L denotes the noise bound after a multiplicative depth of L. We also introduce the expansion factor δ, which bounds the product of two polynomials. For two polynomials A and B, the expansion can be expressed as $\delta = sup\{\|A \cdot B\|_\infty / \|A\|_\infty \|B\|_\infty\} = n$.

4.1.2 FV The noise bound has been thoroughly studied in [19], thus we only recall some key information below.
Initial noise. To determine the initial noise, we apply the decryption procedure on a fresh ciphertext, focusing on the encryption of a 0:

$$C[0] + C[1] \cdot S_{key} = (AS + E)U + E_1 + (AU + E_2)S_{key} = EU + E_1 + E_2 S$$

Thus, the initial noise is $B_0 = B_{err}(1 + 2nB_{key})$.
Multiplicative noise. Following the approach in [19], to ensure concreteness of FV, we must have $C_1^L B_0 + L C_1^{L-1} C_2 < (\Delta - r_t(q))/2$ where $C_1 = \delta t(4 + \delta B_{key})$, $C_2 = \delta^2 B_{key}(B_{key} + t^2) + \delta \omega l_{\omega,q} B_{err}$, $\Delta = \lfloor q/t \rfloor$ and $r_t(q) = q - \Delta t$.

4.1.3 SHIELD The authors of [16] only provided an asymptotic evaluation of SHIELD's noise growth. We develop below a more precise calculation, providing the constant terms. In this section, BD and BDI refer to SHIELD.BD and SHIELD.BDI respectively.

Initial noise. To determine the initial noise, we apply the decryption procedure on a fresh ciphertext, focusing on the encryption of a 0:

$$\begin{aligned}\mathbf{C}\cdot\mathbf{S_{key}} &= (m\cdot \mathrm{BDI}(\mathbf{I_{N\times N}}) + \mathbf{r_{N\times 1}}\cdot\mathbf{P_{key}} + \mathbf{E_{N\times 2}})\cdot\mathbf{S_{key}}\\ &= \mathbf{r_{N\times 1}}\cdot\mathbf{P_{key}}\cdot\mathbf{S_{key}} + \mathbf{E_{N\times 2}}\cdot\mathbf{S_{key}} = \mathbf{r_{N\times 1}}\cdot E + \mathbf{E_{N\times 2}}\cdot\mathbf{S_{key}}\end{aligned}$$

We set $\mathcal{E} = \mathbf{r_{N\times 1}}\cdot E + \mathbf{E_{N\times 2}}\cdot\mathbf{S_{key}}$ and we have

$$\|\mathcal{E}[i]\|_\infty < nB_{err} + B_{err} + n\cdot B_{err}\cdot B_{key} = B_{err}(1+n(1+B_{key}))$$

Thus, the initial noise can be bounded by $B_0 = B_{err}(1+n(1+B_{key}))$.

Multiplicative noise. To determine the noise after an homomorphic multiplication in SHIELD, we apply the decryption procedure after the multiplication step. Recall that SHIELD.Mult$(\mathbf{C_1},\mathbf{C_2}) = \mathrm{BD}(\mathbf{C_1})\cdot\mathbf{C_2}$

$$\begin{aligned}\mathrm{BD}(\mathbf{C_1})\cdot\mathbf{C_2}\cdot\mathbf{S_{key}} &= \mathrm{BD}(\mathbf{C_1})(m_2\ \mathrm{BDI}(\mathbf{I_{N\times N}})\cdot\mathbf{S_{key}} + \mathcal{E}_2)\\ &= m_2\ \cdot\mathrm{BD}(\mathbf{C_1})\cdot\mathrm{BDI}(\mathbf{I_{N\times N}})\cdot\mathbf{S_{key}} + \mathrm{BD}(\mathbf{C_1})\cdot\mathcal{E}_2\\ &= m_2\ \cdot\mathbf{C_1}\cdot\mathbf{S_{key}} + \mathrm{BD}(\mathbf{C_1})\cdot\mathcal{E}_2\\ &= m_1\cdot m_2\cdot\mathrm{BDI}(\mathbf{I_{N\times N}})\cdot\mathbf{S_{key}} + m_2\cdot\mathcal{E}_1 + \mathrm{BD}(\mathbf{C_1})\cdot\mathcal{E}_2\end{aligned}$$

We set $\mathcal{E}_\times = m_2\cdot\mathcal{E}_1 + \mathrm{BD}(\mathbf{C_1})\cdot\mathcal{E}_2$. To bound $\mathcal{E}_\times$, which is a vector, one must bound each elements. $\mathrm{BD}(\mathbf{C_1})$ is always a $N\times N$-matrix of binary polynomials. Thus, each row of $\mathrm{BD}(\mathbf{C_1})\cdot\mathcal{E}_2$ is a product/accumulation of $N = 2\log_2 q$ binary polynomials with polynomials bounded by $\|\mathcal{E}_2[i]\|_\infty$. After one homomorphic multiplication, the noise can be bounded by

$$\|\mathcal{E}_\times[i]\|_\infty < m_2\cdot B_0^{(1)} + (2n\log_2 q)B_0^{(2)} < B_0(1+2n\log_2 q) \tag{1}$$

Then, by an immediate induction, the noise after L homomorphic multiplications can be expressed as $B_L = B_0(1+2n\log_2 q)^L$. To be able to decrypt without error after L homomorphic multiplications, the final noise must be lower than $q/2$. We must have $q/2 > B_0(1+2n\log_2 q)^L$.

Better noise for multiplication. Unlike in FV, noise in SHIELD grows slowly if a ciphertext is multiplied by a fresh one. By carefully examining Equation 1, one can deduce that the noise of each ciphertext is independent. Thus, the multiplicative noise growth can be more finely managed. When a ciphertext is multiplied by L other fresh ciphertexts, the noise growth can be expressed as $B_L = B_0 + L(2n\log_2 q)B_0 = B_0(1+L(2n\log 2q))$.

With batching. Earlier, we extracted noise parameters when $m = 1$. However, if one wants to use batch operations, the message is a polynomial with coefficients in $\{0,1\}$. In that case, noise equation of the optimized circuit becomes $B_{i+1} = n\cdot B_i + (2n\log_2 q)B_0$.

It is an arithmetico-geometric sequence of the form $B_{i+1} = a\cdot B_i + b$, where $a = n$ and $b = 2n\log_2 qB_0$. So $B_L = a^L(B_0 - r) + r$, with $r = \frac{b}{1-a}$.

Table 1: Maximum $\log_2 q$ for a given dimension n, where λ is the security level. $\sigma_{err} = 2\sqrt{n}$.

n	1024	2048	4096	8192
$\lambda = 80$ bits	46 bits	88 bits	174 bits	348 bits
$\lambda = 128$ bits	30 bits	58 bits	112 bits	222 bits

4.2 Security

4.2.1 Attacks As expected in cryptography, all the schemes presented here come with hardness results, provided by reductions to the Ring-LWE problem. Yet, beyond these asymptotic reductions, we need concrete hardness results to choose the scheme parameters according to a security level objective, e.g. 80 bits or 128 bits. Albrecht et al. [3] summarize the state-of-the-art of the attacks against LWE. All of them apply against ring instances which are particular cases. Another line of algebraic attacks exists also against Ring-LWE [23].

In this work, we rely on Albrecht's estimator (available on BitBucket[4]) to estimate attacks performances. It turned out that the recent attack described in [2] was not the best against our instances. For a brief overview we put in Table 1 the maximal allowed $\log q$ for several dimensions at 80 and 128 bits of security.

4.2.2 Determining security parameters Real use-cases of homomorphic cryptography define requirements for the multiplicative depth L and a security level λ to achieve, then one needs to choose the corresponding security parameters.

Upper bound on q. First, one sets an arbitrary (tentative) n, the cyclotomic polynomial degree, as low as possible. Then, with the attack models of the estimator, one can determine an upper-bound of q.

Lower bound on q. The next step is to evaluate if such a modulus q is compatible with the required multiplicative depth L. This depends of the scheme, unlike the upper bound. If it does not, *i.e.* the security requires a q smaller than what is needed by the multiplicative depth, one must increase n and go back to the previous step in order to attempt to solve again the two inequalities on q.

We summarize the approach in Algorithm 1, which we used to compute the tables presented below.

5 Practical parameters

In this section, we explore different settings: arbitrary circuit, optimized circuit, NWC, batching, and report concrete parameters for scheme comparison.

[4] `https://bitbucket.org/malb/lwe-estimator`, commit `eb45a74`

Algorithm 1 Determine (n, σ and q) parameters from (L, λ) for a given scheme

```
1: function ChooseParam(scheme, L, λ)
2:     q ← 0
3:     n ← 1
4:     repeat
5:         σ ← 2√n
6:         M_q ← Max-modulus(n, λ)
7:         m_q ← Min-modulus(n, L, scheme)
8:         if m_q < M_q then
9:             q ← m_q
10:        else
11:            n ← n + 1
12:        end if
13:    until q ≠ 0
14:    return n, σ, q
15: end function
```

5.1 Multiplicative depth for an arbitrary binary circuit

Table 2 provides parameters for FV and SHIELD for 80 and 128 bits of security. They are extracted in the proved-hardness regime, that is to say $\sigma_{err} = 2\sqrt{n}$ for each scheme

Values for SHIELD seem the best in the tables. However the number of sub-polynomials for a given ciphertext explodes because it is proportional to $\log_2 q$ for SHIELD. For example, with $L = 5$, a ciphertext in SHIELD contains $2 \times N = 4 \times \log_2 q = 480$ sub-polynomials of degree-2793 with 120 bits coefficients, whereas FV only requires two sub-polynomials of degree-3731 with 159 bits coefficients.

Consequently, in the case of an arbitrary binary circuit, FV is best.

5.2 Multiplicative depth for an optimized circuit

As stated in the previous section, SHIELD seems inefficient for arbitrary circuits. However, they have a really interesting feature: when a ciphertext is multiplied by a fresh ciphertext, the noise growth is additive instead of multiplicative for binary messages. Table 3 provides parameters for SHIELD for this optimized circuit. FV is omitted here, because it presents no particular optimization.

Results are very impressive, SHIELD scale to large multiplicative degree with nearly no impact on n and q. For SHIELD and for 80 bits of security, the modulus only increases by 5 bits between a multiplicative depth of 1 and 20 when the degree of the associated cyclotomic polynomial remains under 1024. As a reminder from Table 2, FV requires at least $n = 14347$ and $\log_2 q = 611$ bits for a multiplicative depth of 20.

SHIELD is clearly better than FV in this setting, which is not about evaluating circuit of depth L for *all* inputs, yet still a degree-L function.

Table 2: Parameters for FV, SHIELD and F-NTRU, where λ is the security level and L the multiplicative depth. Arbitrary circuit.

(a) Selection of parameters for FV. Binary key, $\sigma_{err} = 2\sqrt{n}$.

L	$\lambda = 80$ bits				$\lambda = 128$ bits			
	$\omega = 32$ bits		$\omega = 64$ bits		$\omega = 32$ bits		$\omega = 64$ bits	
	$\log_2 q$	n	$\log_2 q$	n	$\log_2 q$	n	$\log_2 q$	n
1	54	1214	87	2008	55	1940	88	3156
5	159	3731	193	4540	166	6060	200	7321
10	303	7134	337	7964	317	11675	351	12844
15	454	10719	488	11458	475	17452	509	18722
20	611	14347	645	15127	639	23417	673	24709

(b) Selection of parameters for SHIELD. Binary error, $\sigma_{key} = 2\sqrt{n}$.

L	$\lambda = 80$ bits		$\lambda = 128$ bits	
	$\log_2 q$	n	$\log_2 q$	n
1	36	784	38	1306
5	120	2793	124	4496
10	238	5595	247	9021
15	364	8582	376	13820
20	495	11609	511	18799

5.3 The case of the Negative Wrapped Convolution

Attracted by its performance, a majority of polynomial multiplication implementations use the NWC NTT. We provide in Table 4 the associated parameters for FV. For SHIELD, parameters seem quite independent of the multiplicative depth. Because the polynomial degree is oversized due to NWC, a security of $\lambda = 80$ bits requires $n = 1024$, cf Table 3, and we can then go to very high L. Similarly, $n = 2048$ is required for $\lambda = 128$ bits. For the same use case as FV for $\lambda = 80$ bits, the polynomial degree is always 1024, with $\log_2 q = 39$ bits for a multiplicative degree of 5, 40 bits for a multiplicative degree of 10, and 41 bits for a multiplicative depth of 20. As a reminder, NWC uses the cyclotomic polynomial $x^n + 1$ and the NTT computations are performed in the ring $\mathbb{Z}[x]/(x^n + 1)$. Hence the polynomial reduction is directly integrated into NTT computations. This performance tweak comes at the cost of disabling the packing of several messages into one ciphertext, no batching possible. Parameters are selected to maximize the multiplicative depth for a given n, which is necessarily a power of 2, because the NWC NTT set the cyclotomic polynomial to $x^n + 1$. When compared to the previous case, this slightly increases the size of the modulus, for a given multiplicative depth. For example with FV, for a multiplicative depth of 4, optimized parameters are $n = 3084$ and $\log_2 q = 132$. In a NWC NTT scenario,

Table 3: Parameters for SHIELD and F-NTRU, where λ is the security level and L the multiplicative degree. Optimized circuit. Binary message (No batching).

(a) Selection of parameters for SHIELD. Binary error, $\sigma_{key} = 2\sqrt{n}$.

L	$\lambda = 80$ bits		$\lambda = 128$ bits	
	$\log_2 q$	n	$\log_2 q$	n
1	36	784	38	1306
5	39	857	41	1411
10	40	879	42	1447
15	40	879	42	1448
20	41	901	43	1492

Table 4: Parameters for FV for the NWC NTT, where λ is the security level and L the multiplicative depth. Binary key, $\sigma_{err} = 2\sqrt{n}$. Warning: no batching with the NWC NTT.

n	$\lambda = 80$ bits				$\lambda = 128$ bits			
	$\omega = 32$ bits		$\omega = 64$ bits		$\omega = 32$ bits		$\omega = 64$ bits	
	$\log_2 q$	L	$\log_2 q$	L	$\log_2 q$	L	$\log_2 q$	L
1024	×	×	×	×	×	×	×	×
2048	80	2	87	1	55	1	×	×
4096	161	5	166	4	109	3	89	1
8192	334	11	338	10	198	6	202	5
16384	677	22	679	21	443	14	445	13

new parameters are $n = 4096$ and $\log_2 q = 135$ bits. Thus, the ciphertexts are slightly larger when compared to optimized ones, but the computation time is still better than for standard multiplication which requires a $2n$-NTT with zero padding.

5.4 The impact of batching

As stated in Section 3.3, the batching technique is very useful to reduce the ciphertext expansion. Table 5 provides parameters for FV and SHIELD when the batching technique is used, in an optimized circuit as described in Section 5.2. Unlike when the messages are binary, SHIELD parameters becomes sensitive to the multiplicative depth.

As early as a depth of 2, the dimension goes over 1024 and implies an associated NTT of size 2048. Moreover, the modulus q grows significantly with the depth, on average 12 more bits per level. which leads to more and more sub-polynomials for a given ciphertext. For a multiplicative depth of 10, SHIELD with batching requires 596 sub-polynomials of degree 3487 with coefficients of 149 bits, while without batching it only requires 160 sub-polynomials of degree 879 with coefficients of 40 bits.

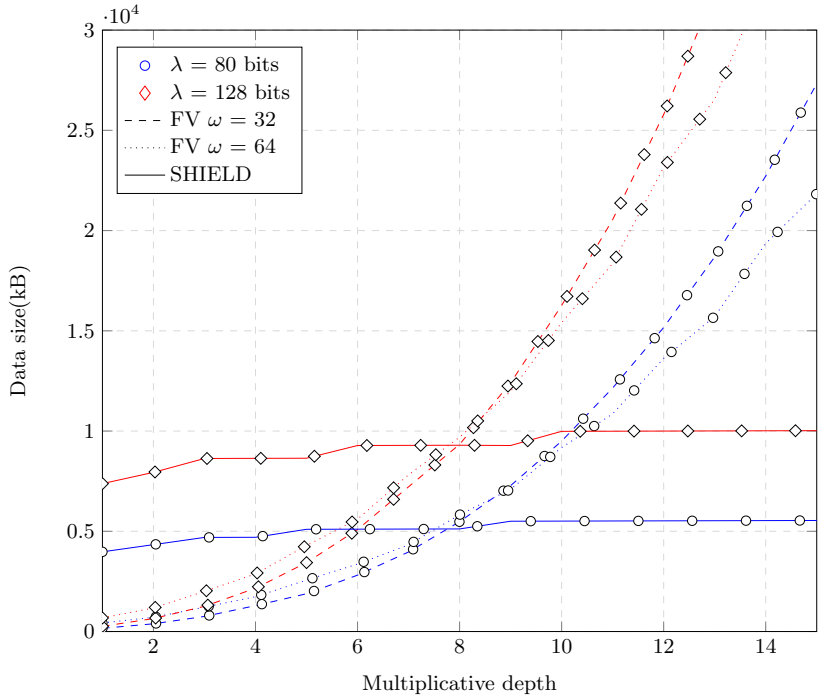

Fig. 1: Data size for an homomorphic scenario with 8 encryptions.

As we see here, batching in FV has no significant impact on the parameters, whereas it is the opposite for SHIELD.

5.5 Keys and ciphertexts sizes

Figure 6 provides the volume of data for FV and SHIELD in a scenario requiring 8 bits of information. For SHIELD, it is only the size of 8 ciphertexts. For FV, the size of relinearization keys are also included because they are required during the homomorphic multiplication. For small multiplicative depths, namely under 8, FV requires a lower amount of data than SHIELD. But for larger depths, the improved noise management of SHIELD is highly beneficial. The main issue for FV is the size of the relinearization key. For a multiplicative depth of 15, it is as large as 17.4 MB, when SHIELD does not require such a key. It can be reduced a bit by enlarging ω at an additional computation cost. However SHIELD is no longer the lightest with batching. Even a packing of only 3 accounts for the same as what we observe in FV with increase in the multiplicative depth.

6 Conclusion

This study has provided some new and helpful information concerning practical issues of homomorphic encryption. Two different schemes have been studied: FV,

a second, and SHIELD a third generation scheme. FV has in major cases shorter ciphertexts than the others, thanks to the relinearization step. More precisely, an FV ciphertext is only composed of two polynomials, but with higher degree and coefficient size. Also, FV is very sensitive to the multiplicative depth and has no particular optimization for any binary circuit. SHIELD is a third generation scheme, which means that the relinearization step is somehow included in the multiplication. The noise growth is much lower than for FV, leading to ciphertexts composed of smaller sub-polynomials. Yet there are many polynomials to handle, $2 \times \log_2 q$ times more. This is not a major issue for SHIELD, because if the computation is optimized to prefer multiplication with fresh ciphertexts, it can achieve a very high multiplicative depth (up to 20) without impacting much the sub-polynomial size. For example, maintaining it below 1024 for $\log_2 q \leq 41$ bits. As SHIELD authors reported, numerous but small polynomials multiplication can be very efficiently implemented in GPU and counterbalance the size of ciphertexts.

About batching: unlike FV, SHIELD is very sensitive to batching. For a multiplicative depth of 4, SHIELD with batching requires $n = 1649$ and $\log_2 q = 72$. This has a critical impact compared to the no-batching version because we now require to double the size of the NTT/NWC, and double the size of the integer multiplication operands. And this phenomenon worsens when the multiplicative depth grows.
To conclude, SHIELD is a good candidate when the multiplicative depth is important, namely $L \geq 10$. But this only holds when the bandwidth is not such a problem. However, if one wants to efficiently use the bandwidth, if the multiplicative depth is not too important ($L \leq 9$), then FV is probably a better solution, and even more so when coupled with the batching technique.

Further work on implementations will provide even better insights on the real performances and behaviours of these schemes.

References

1. Martin Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched ntru assumptions: Cryptanalysis of some fhe and graded encoding schemes. Cryptology ePrint Archive, Report 2016/127, 2016.
2. Martin R. Albrecht. On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. Cryptology ePrint Archive, Report 2017/047, 2017. `http://eprint.iacr.org/2017/047`.
3. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. Cryptology ePrint Archive, Report 2015/046, 2015.
4. Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Proc. of Cryptography and Coding: 14th IMA International Conference – IMACC 2013*, pages 45–64. Springer, 2013.
5. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012*, pages 868–886. Springer, 2012.

6. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proc. of the 3rd Innovations in Theoretical Computer Science Conference – ITCS 2012*, pages 309–325. ACM, 2012.
7. Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *Proc. of the 5th Conference on Innovations in Theoretical Computer Science – ITCS 2014*, pages 1–12. ACM, 2014.
8. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 3–33. Springer, 2016.
9. Yarkın Doröz and Berk Sunar. Flattening ntru for evaluation key free homomorphic encryption. Cryptology ePrint Archive, Report 2016/315, 2016.
10. Léo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.
11. Junfeng Fan and Frederick Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012.
12. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.
13. Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.
14. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Proc. of Advances in Cryptology – CRYPTO 2013*, pages 75–92, 2013.
15. Shai Halevi. Helib. Available at `https://github.com/shaih/HElib`.
16. Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. Shield: Scalable homomorphic implementation of encrypted data-classifiers. *Accepted to IEEE Transactions on Computers*, 2015.
17. Paul Kirchner and Pierre-Alain Fouque. Comparison between Subfield and Straightforward Attacks on NTRU. Cryptology ePrint Archive, 2016/717, 2016.
18. Kim Laine, Hao Chen, and Rachel Player. Simple encrypted arithmetic library - seal (v2.1). Technical report, September 2016.
19. Tancrede Lepoint and Michael Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Proc. of AFRICACRYPT 2014*, volume 8469 of *LNCS*, pages 318–335, 2014.
20. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Advances in Cryptology–EUROCRYPT 2010*, pages 1–23, 2010.
21. Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *Annual Cryptology Conference*, pages 138–154. Springer, 2010.
22. Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. of Advances in Cryptology — EUROCRYPT 1999*, number 1592 in LNCS, pages 223–238, 1999.
23. Chris Peikert. How (Not) to Instantiate Ring-LWE. Cryptology ePrint Archive, Report 2016/351, 2016.
24. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.

Table 5: Parameters of FV and SHIELD for 80 bits of security when batching is enabled, where L is the multiplicative depth, batching the number of packed operations, m the rank of the cyclotomic polynomial and hw the hamming weight of the associated cyclotomic polynomial. Binary key, $\sigma_{err} = 2\sqrt{n}$

(a) Values for FV. $\omega = 32$ bits.

L	batching	hw	n	m
1	12	33	1296	2835
	72	97	1296	1971
2	2	7	1944	3645
	6	17	1944	3159
	18	49	1944	2997
	20	57	2000	4125
3	4	23	2592	4131
	12	57	2520	4851
	24	59	2592	5265
4	2	7	3240	6075
	4	31	3120	4225
	12	57	3240	6237
	16	73	3200	6375
5	12	33	3888	8505
	24	59	3744	7605
	72	97	3888	5913
6	2	39	4536	7047
	20	57	4400	9075
7	2	41	5060	5819
8	2	7	5832	10935
	6	17	5832	9477
	18	49	5832	8991

(b) Values for SHIELD.

L	batching	hw	n	m
1	6	9	972	1701
	18	25	972	1539
	24	59	864	1755
	40	65	800	1025
2	2	7	1080	2025
	4	31	1200	1625
	8	41	1088	1445
	14	49	1176	1421
	20	57	1200	2475
	60	73	1200	2325
3	2	31	1368	1805
	24	59	1440	2925
4	2	7	1800	3375
	6	9	1764	3087
	42	73	1764	2107
5	2	7	2058	2401
	6	17	1944	3159
	18	49	1944	2997
	20	57	2000	4125
6	6	9	2268	3969
	22	41	2420	2783
7	4	23	2592	4131
	24	59	2592	5265
8	2	7	3000	5625
	6	9	2916	5103
	18	25	2916	4617
	30	49	3000	3875
	36	99	3024	5733

Table 6: Parameters size for FV and SHIELD, where λ is the security level and L the multiplicative depth. Binary key, $\sigma_{err} = 2\sqrt{n}$.

(a) FV

L	$\lambda = 80$ bits				$\lambda = 128$ bits			
	$\omega = 32$ bits		$\omega = 64$ bits		$\omega = 32$ bits		$\omega = 64$ bits	
	C, P_{key}	γ	C, P_{key}	γ	C, P_{key}	γ	C, P_{key}	γ
1	16.0 KB	32.0 KB	42.6 KB	85.3 KB	26.0 KB	52.1 KB	67.8 KB	135.6 KB
5	144.8 KB	724.2 KB	213.9 KB	855.7 KB	245.6 KB	1.4 MB	357.5 KB	1.4 MB
10	527.7 KB	5.2 MB	655.2 KB	3.8 MB	903.6 KB	8.8 MB	1.1 MB	6.4 MB
15	1.1 MB	17.4 MB	1.3 MB	10.7 MB	2.0 MB	29.6 MB	2.3 MB	18.2 MB
20	2.1 MB	41.8 MB	2.3 MB	25.6 MB	3.6 MB	71.4 MB	4.0 MB	43.6 MB

(b) SHIELD. Optimized circuit.

L	$\lambda = 80$ bits				$\lambda = 128$ bits			
	No batching		Batching		No batching		Batching	
	C	P_{key}	C	P_{key}	C	P_{key}	C	P_{key}
1	496.1 KB	6.9 KB	375 KB	5.4 KB	920.8 KB	12.1 KB	521.4 KB	7.2 KB
5	636.5 KB	8.2 KB	5.2 MB	32.3 KB	1.1 MB	14.1 KB	7.4 MB	44.3 KB
10	686.7 KB	8.6 KB	30 MB	105.1 KB	1.2 MB	14.8 KB	43 MB	145.9 KB
15	686.7 KB	8.6 KB	94.8 MB	226.8 KB	1.2 MB	14.8 KB	136.6 MB	316.5 KB
20	739.5 KB	9.0 KB	223 MB	402 KB	1.3 MB	15.7 KB	320 MB	559.2 KB