



HAL
open science

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

Vincent Migliore, Guillaume Bonnoron, Caroline Fontaine

► To cite this version:

Vincent Migliore, Guillaume Bonnoron, Caroline Fontaine. Determination and exploration of practical parameters for the latest Somewhat Homomorphic Encryption (SHE) Schemes. 2016. hal-01394362v2

HAL Id: hal-01394362

<https://hal.science/hal-01394362v2>

Preprint submitted on 11 Jan 2017 (v2), last revised 14 Apr 2017 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

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

Vincent Migliore¹, Guillaume Bonnoron², Caroline Fontaine³

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@telecom-bretagne.eu`

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 [?] or addition [16]. But it was only with the works from [15] and [10] that key ideas were introduced to support both operations simultaneously. These schemes have been followed by many others [3,8,5,4,9,11,6,12]. 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 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, HE theoretical background has been evolving. Thus, it has remained a real challenge to draw practical parameters. Moreover, former publications usually present values for specific use-cases and do not address 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 three of most promising schemes: FV[9], SHIELD[12] and F-NTRU[8]. Usually, implementations target YASHE'[3] instead of FV. However confidence in this scheme has been recently damaged by the subfield/sublattice attack [1,?]. Making YASHE' immune to these attacks would lead to oversize its parameters, far too much for practical use [?]. We selected SHIELD and F-NTRU for comparison because they are the 3^{rd} generation schemes equivalent to, respectively, FV and YASHE'. With larger costs for the first homomorphic multiplications, these schemes have a much better asymptotic behavior. Thus, even if F-NTRU is based on YASHE', secure parameters can still be selected. Another interesting candidate would have been BGV[5], used in HELib [?]. However BGV could not be fairly compared because of the modulus-switching operations that prevent efficient hardware implementations.

The main contributions of the paper are: a concise presentation of the three schemes with harmonized notation; a review of parameters extraction for each 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, SHIELD and F-NTRU 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 $\mathbb{Z}_q[X] = \mathbb{Z}[X]/q\mathbb{Z}$ 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 $\text{round}(a)$ 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: $l = \log_2 q$, $N = 2l$ 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 [14].

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 \mathbf{s} from pairs (A, C) sampled from $L_{S,\chi}$.

The hardness of Ring-LWE problem depend 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 [9] is a transposition of the scale-invariant Brakerski scheme [4] 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 $t \in (1, q)$ 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 bits or 64 bits integer for computational aspects.

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

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

- **FV.GenKeys** (λ) :
 - $S \leftarrow D_{R_q, \sigma_{key}}, A \leftarrow U_{R_q}, E \leftarrow D_{R_q, \sigma_{err}}$
 - $P_{key} = (-AS + E, A)$
 - $S_{key} = S$
 - return (P_{key}, S_{key})
- **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
- **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
- **FV.Decrypt** (C, S_{key}) :
 - $\widetilde{M} = [C[0] + C[1]S_{key}]_q$
 - $m = \left[\frac{t}{q}\widetilde{M}[0] \right]$
 - return m
- **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_+
- **FV.Mult** (C_A, C_B, γ) :
 - $\widetilde{C}_0 = \left[\left[\frac{t}{q}C_A[0] \times C_B[0] \right] \right]_q$
 - $\widetilde{C}_1 = \left[\left[\frac{t}{q}(C_A[0] \times C_B[1] + C_A[1] \times C_B[0]) \right] \right]_q$

$$\tilde{C}_2 = \left[\left[\frac{t}{q} C_A[1] \times C_B[1] \right] \right]_q$$

$$C_\times = \text{FV.Relin}(\tilde{C}_0, \tilde{C}_1, \tilde{C}_2, \gamma)$$

return C_\times

– **FV.Relin**($\tilde{C}_0, \tilde{C}_1, \tilde{C}_2, \gamma$) :

$$C_R = (C_{R,0}, C_{R,1})$$

$$C_{R,0} = \left[\tilde{C}_0 + \left\langle \text{FV.WordDecomp}_{w,q}(\tilde{C}_2), \gamma[0] \right\rangle \right]_q$$

$$C_{R,1} = \left[\tilde{C}_1 + \left\langle \text{FV.WordDecomp}_{w,q}(\tilde{C}_2), \gamma[1] \right\rangle \right]_q$$

return C_R

3.2 SHIELD

SHIELD [12] is a transposition of GSW scheme [11] 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.

– **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}

– **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}) :
 $\mathbf{r}_{N \times 1} \leftarrow B_{R_q}^{N \times 1}, \mathbf{E}_{N \times 2} \leftarrow D_{R_q, \sigma_{err}}^{N \times 2}$
 $\mathbf{C} = \mathbf{C}_{N \times 2} = m \cdot \text{BDI}(\mathbf{I}_{N \times N}) + \mathbf{r}_{N \times 1} \cdot \mathbf{P}_{key} + \mathbf{E}_{N \times 2}$
return \mathbf{C}
- **SHIELD.Decrypt**($\mathbf{C}, \mathbf{S}_{key}$) :
 $\mathbf{M} = \mathbf{C} \cdot \mathbf{S}_{key} = m \cdot \text{BDI}(\mathbf{I}_{N \times N}) \cdot \mathbf{S}_{key} + error$
 $m = \left\lfloor \frac{2}{q} \cdot \mathbf{M}[0][0] \right\rfloor$
return m
- **SHIELD.Add**($\mathbf{C}_1, \mathbf{C}_2$) :
 $\mathbf{C}_+ = \mathbf{C}_1 + \mathbf{C}_2$
return \mathbf{C}_+
- **SHIELD.Mult**($\mathbf{C}_1, \mathbf{C}_2$) :
 $\mathbf{C}_\times = \text{BD}(\mathbf{C}_1) \cdot \mathbf{C}_2$
return \mathbf{C}_\times

3.3 F-NTRU

F-NTRU [8] scheme is the latest homomorphic encryption scheme presented in this paper. In contrast to FV or SHIELD, the decryption is not based on a polynomial pair where one member contains a noisy image of the secret key to help decryption. Instead, F-NTRU is based on LTV scheme, which requires an invertible polynomial as a key to avoid the polynomial-pair requirement. Even if F-NTRU and YASHE' share the same problematic security assumptions, F-NTRU adapts latest noise management techniques proposed in GSW to reduce the noise growth. This allows for secure parameters even under the subfield/sublattice attacks. This new noise management requires an additional operation called FLATTEN, that requires an array of polynomials instead of a single one like in YASHE'.

- **F-NTRU.BDI**(\mathbf{A}) :
 $(\mathbf{A} \in B_{R_q}^{l \times l})$
 $\mathbf{B} \in B_{R_q}^{l \times 1}$
for $i = 0$ to $l - 1$
 $\mathbf{B}[i] = \sum_{j=0}^{l-1} \mathbf{A}[i][j] 2^j$
end for
return \mathbf{B}
- **F-NTRU.BD**(\mathbf{A}) :
 $(\mathbf{A} \in B_{R_q}^{l \times 1})$
 $\mathbf{B} \in B_{R_q}^{l \times l}$
for $i = 0$ to $l - 1$
for $j = 0$ to $l - 1$
 $\mathbf{B}[i][j] = \mathbf{A}[i][j]$
end for
end for
return \mathbf{B}
- **F-NTRU.FLATTEN**(\mathbf{A}) : $(\mathbf{A} \in B_{R_q}^{l \times l})$
 $\mathbf{B} \in B_{R_q}^{l \times l}$
 $\mathbf{B} = \mathbf{F-NTRU.BD}(\mathbf{F-NTRU.BDI}(\mathbf{A}))$
return \mathbf{A}
- **F-NTRU.GenKeys**(λ) :
 $G \leftarrow D_{R_q, \sigma_{key}}, F' \leftarrow D_{R_q, \sigma_{key}}$
 $B = A \cdot T + E$
 $S_{key} = F = 2F' + 1$
 $P_{key} = 2GF^{-1}$
return (P_{key}, S_{key})
- **F-NTRU.Encrypt**(m, P_{key}) :
 $\mathbf{S} \leftarrow D_{R_q, \sigma_{err}}^{l \times 1}, \mathbf{E} \leftarrow D_{R_q, \sigma_{err}}^{l \times 1}$
for $i = 0$ to $l - 1$
 $\mathbf{C}'_{l \times 1}[i] = P_{key} \cdot \mathbf{S}[i] + 2 \mathbf{E}[i] + 0$
end for
 $\mathbf{C} = \mathbf{F-NTRU.FLATTEN}(m \cdot \mathbf{I}_{l \times l} + \text{BD}(\mathbf{C}'_{l \times 1}))$
- **F-NTRU.Decrypt**(C, S_{key}) :
 $c_0 = \text{BDI}(C_{(0, l-1)}, \dots, C_{(0, 0)})$
 $m = \lfloor c_0 S_{key} \rfloor \text{mod} 2$
return m
- **F-NTRU.Add**($\mathbf{C}_1, \mathbf{C}_2$) :
 $\mathbf{C}_+ = \mathbf{F-NTRU.FLATTEN}(\mathbf{C}_1 + \mathbf{C}_2)$
return \mathbf{C}_+
- **F-NTRU.Mult**($\mathbf{C}_1, \mathbf{C}_2$) :
 $\mathbf{C}_\times = \mathbf{F-NTRU.FLATTEN}(\mathbf{C}_1 \cdot \mathbf{C}_2)$
return \mathbf{C}_\times

3.4 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 [18], 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.

3.5 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 3.4. Thus, the NWC is optimized for performance but incompatible with batching techniques.

4 Parameters extraction

As described in Section 3.4, 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 $\|A\|_\infty = \max_{0 \leq i < n} |a_i|$. 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 [13], 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:

$$C[0] + C[1] \cdot S_{key} = (AS + E)U + E_1 + (AU + E_2)S_{key} = EU + E_1 + E_2S$$

Thus, the initial noise is $B_0 = B_{err}(1 + 2nB_{key})$.

Multiplicative noise. Following the approach in [13], to ensure concreteness of FV, we must have $C_1^L B_0 + LC_1^{L-1} C_2 < (\Delta - r_t(q))/2$ 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 Authors of [12] 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** variants.

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

$$\begin{aligned} \mathbf{C} \cdot \mathbf{S}_{key} &= (m \cdot \mathbf{BDI}(\mathbf{I}_{N \times N}) + \mathbf{r}_{N \times 1} \cdot \mathbf{P}_{key} + \mathbf{E}_{N \times 2}) \cdot \mathbf{S}_{key} \\ &= \mathbf{r}_{N \times 1} \cdot \mathbf{P}_{key} \cdot \mathbf{S}_{key} + \mathbf{E}_{N \times 2} \cdot \mathbf{S}_{key} = \mathbf{r}_{N \times 1} \cdot E + \mathbf{E}_{N \times 2} \cdot \mathbf{S}_{key} \end{aligned}$$

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 $\text{SHIELD.Mult}(\mathbf{C}_1, \mathbf{C}_2) = \text{BD}(\mathbf{C}_1) \cdot \mathbf{C}_2$

$$\begin{aligned} \text{BD}(\mathbf{C}_1) \cdot \mathbf{C}_2 \cdot \mathbf{S}_{key} &= \text{BD}(\mathbf{C}_1)(m_2 \mathbf{BDI}(\mathbf{I}_{N \times N}) \cdot \mathbf{S}_{key} + \mathcal{E}_2) \\ &= m_2 \cdot \text{BD}(\mathbf{C}_1) \cdot \mathbf{BDI}(\mathbf{I}_{N \times N}) \cdot \mathbf{S}_{key} + \text{BD}(\mathbf{C}_1) \cdot \mathcal{E}_2 \\ &= m_2 \cdot \mathbf{C}_1 \cdot \mathbf{S}_{key} + \text{BD}(\mathbf{C}_1) \cdot \mathcal{E}_2 \\ &= m_1 \cdot m_2 \cdot \mathbf{BDI}(\mathbf{I}_{N \times N}) \cdot \mathbf{S}_{key} + m_2 \cdot \mathcal{E}_1 + \text{BD}(\mathbf{C}_1) \cdot \mathcal{E}_2 \end{aligned}$$

We set $\mathcal{E}_\times = m_2 \cdot \mathcal{E}_1 + \text{BD}(\mathbf{C}_1) \cdot \mathcal{E}_2$. To bound \mathcal{E}_\times , which is a vector, one must bound each elements. $\text{BD}(\mathbf{C}_1)$ is always a $N \times N$ -matrix of binary polynomials. Thus, each row of $\text{BD}(\mathbf{C}_1) \cdot \mathcal{E}_2$ 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

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

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 $B_L = B_0 + L(2n \log_2 q)B_0 = B_0(1 + L(2n \log_2 q))$.

With batching. Earlier, we extracted noise parameters when $m = \tilde{m} = 1$. 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 $b = 2n \log_2 q B_0$. So $B_L = a^L(B_0 - r) + r$, with $r = \frac{b}{1-a}$.

4.1.4 F-NTRU Authors of F-NTRU also precisely analyzed the noise growth, but the study was done for integer messages. In the following, we adapt their equations to binary messages. In this section, **BD**, **BDI** and **FLATTEN** refer to F-NTRU variants.

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

$$\begin{aligned} \mathbf{BDI}(\mathbf{C}) \cdot \mathbf{S}_{\mathbf{key}} &= \mathbf{BDI}(\mathbf{FLATTEN}(\mathbf{BD}(\mathbf{C}'_{1 \times 1}))) \cdot \mathbf{S}_{\mathbf{key}} \\ &= \mathbf{BDI}(\mathbf{BD}(\mathbf{BDI}(\mathbf{BD}(\mathbf{C}'_{1 \times 1})))) \cdot \mathbf{S}_{\mathbf{key}} = \mathbf{C}'_{1 \times 1} \cdot \mathbf{S}_{\mathbf{key}} \end{aligned}$$

Thus, the initial noise can be expressed as

$$\begin{aligned} \|(\mathbf{C}'_{1 \times 1} \cdot \mathbf{S}_{\mathbf{key}})[i]\|_{\infty} &\leq \|P_{key} \cdot \mathbf{S}[i] \cdot S_{key}\|_{\infty} + \|2\mathbf{E}[i] \cdot S_{key}\|_{\infty} \\ &= \|2GF^{-1} \cdot \mathbf{S}[i] \cdot F\|_{\infty} + \|2\mathbf{E}[i] \cdot (2F' + 1)\|_{\infty} \\ &= \|2G \cdot \mathbf{S}[i]\|_{\infty} + \|2\mathbf{E}[i] \cdot (2F' + 1)\|_{\infty} \end{aligned}$$

Since $\|G\|_{\infty} = \|F'\|_{\infty} = B_{key}$, $\|\mathbf{S}[i]\|_{\infty} = \|\mathbf{E}[i]\|_{\infty} = B_{err}$, we have

$$B_0 \leq 2B_{err}(3nB_{key} + 1)$$

Multiplicative noise. In F-NTRU, a ciphertext is a $l \times l$ -matrix of degree- n binary polynomials. As proposed in [8], in order to reduce the number of sub-polynomials for the homomorphic multiplication, one can apply a word decomposition instead of a bit decomposition in **BD**/**BDI**. Following the same notation than FV, polynomials are split with segments of ω bits. However, the reduction of the number of polynomials increases the size of coefficients and thus impact the noise growth. The optimization relies on the following assertion:

$$\mathbf{PowerOf}_{w,q}(\mathbf{WordDecomp}_{w,q}(A) \cdot \mathbf{WordDecomp}_{w,q}(B)) = \mathbf{WordDecomp}_{w,q}(A) \cdot B$$

For c and \tilde{c} two ciphertexts, c' the resulting ciphertext after the homomorphic multiplication, $c_i^{(k)}$ the i^{th} row of c after k homomorphic multiplications, and $c_{(i,j)}$ the i^{th} row of the j^{th} element of $\text{WordDecomp}_{w,q}(c)$, c'_j can be expressed as follows (see [8] for more details)

$$c_j^{(i)} = \sum_{k=0}^{l_{w,q}-1} c_{(j,k)} \cdot \tilde{c}_k^{(i-1)} + c_j^{(i-1)} \tilde{m} + \tilde{c}_j^{(i-1)} m + 2^j m \tilde{m}$$

We set $\|c_j^{(i)}\|_\infty = \|y_i\|_\infty$ and $\|c_{j,k}^{(i)}\|_\infty = \|y_T\|_\infty = \omega$. Then, the first row can be written $y_i = l_{w,q} \cdot \tilde{y}_{i-1} \cdot y_T + y_{i-1} \tilde{m} + \tilde{y}_{i-1} m + m \tilde{m}$.

If we consider binary messages ($m = \tilde{m} \in \{0, 1\}$), with an equivalent noise for \tilde{y}_{i-1} and y_{i-1} , the equation becomes

$$\begin{aligned} \|F \cdot y_i\|_\infty &\leq l_{w,q} \|F \cdot y_{i-1} \cdot y_T\|_\infty + 2 \|F \cdot y_{i-1}\|_\infty + \|F\|_\infty \\ &= l_{w,q} n \omega \|F \cdot y_{i-1}\|_\infty + 2 \|F \cdot y_{i-1}\|_\infty + \|F\|_\infty \end{aligned} \quad (2)$$

Thus, the noise can be expressed as $B_{i+1} \leq (n \cdot l_{w,q} \cdot w + 2) B_i + 2 B_{key} + 1$. It is an arithmetico-geometric sequence of the form $B_{i+1} = a \cdot B_i + b$ where $a = 2 + n \cdot l_{w,q} \cdot w$ and $b = 2 \cdot B_{key} + 1$.

So $B_L = a^L (B_0 - r) + r$, with $r = \frac{b}{1-a} = -\frac{2B_{key}+1}{n \cdot l_{w,q} \cdot w + 1}$.

Better noise for multiplication. Like in SHIELD, when a ciphertext is multiplied by a fresh one, the noise growth is lower. By considering Equation 2 with $\|\tilde{y}_{i-1}\|_\infty = B_0$, the new noise growth can be expressed as

$$\begin{aligned} B_i &= l_{w,q} \cdot n \cdot B_0 \cdot \omega + B_{i-1} + B_0 + 2 B_{key} + 1 \\ B_L &\leq L \cdot (B_0 \cdot (1 + l_{w,q} \cdot n \cdot \omega) + 2 B_{key} + 1) + B_0 \end{aligned}$$

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. [2] 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 [17].

4.2.2 Ring-LWE A common approach to determine the security parameters is to consider the advantage of the attacker at distinguishing Ring-LWE samples from uniformly random samples, *i.e.* breaking decision-Ring-LWE.

For a Ring-LWE sample $(a, u) = (a, as + e)$, the attack consists in finding a short vector $v \in q \cdot \Lambda(a)^\times$, where $\Lambda(a)^\times$ is the dual lattice generated by a . With such a vector, the inner product $\langle v, u \rangle$ gives $\langle v, e \rangle$, which is a small Gaussian. In the case where (a, u) is uniformly random, the inner product is also uniformly

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	54 bits	103 bits	201 bits	401 bits
$\lambda = 128$ bits	44 bits	81 bits	156 bits	307 bits

random, hence the distinction objective. For more information, the reader can refer to [2, Section 5.3].

Thus, the extraction of v is a turning point of the attack. To our knowledge, the best way to find such a short vector is to use the BKZ-2.0 algorithm. The size of the smallest short vector one can recover is linked a parameter called root Hermite factor γ . It captures the quality of the output of BKZ algorithm, the smaller γ , the better the quality. Chen and Nguyen [7] experimented with BKZ and provide time estimates to achieve root Hermite factors. So, following the work in [13], we get a minimal γ from a security objective. Then we get an upper bound on q

$$\log_2 q \leq \min_{m>n} \frac{m^2 \log_2 \gamma(m, \lambda) + m \log_2(\sigma/\alpha)}{m - n} \quad (3)$$

Where σ is the width parameter of the error term, $\alpha = \sqrt{-\log \epsilon/\pi} = 3.7577$ with $\epsilon = 2^{-64}$ the distinguishing advantage of the attacker.

4.2.3 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.

Getting γ . Depending on the security level, one must select the appropriate root Hermite factor γ . Since γ also depends on the dimension m , we provide the following modeling, based on a logarithmic approximation of $\gamma(m)$ for different security level. It follows from the study in [13].

- For $\lambda = 80$ bits, $\gamma(m) = 0.0005649115 \cdot \log_{10}(m) + 1.005907$
- For $\lambda = 128$ bits, $\gamma(m) = 0.0002924305 \cdot \log_{10}(m) + 1.005042$

Upper bound on q . Next, one set an arbitrary (tentative) n , the cyclotomic polynomial degree, as low as possible. Then, with the help of equation 3, one can determine an upper-bound of q .

Lower bound on q . The last 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 increment n and go back to the previous step in order to attempt to solve again the two inequalities on q .

All the values we report in the tables have been determined following Algorithm 1 and are no more optimistic than those from estimators in [2, Table 2].

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

```

1: function CHOOSEPARAM(scheme,  $L, \lambda$ )
2:    $q \leftarrow 0$ 
3:    $n \leftarrow 1$ 
4:   repeat
5:      $\sigma \leftarrow 2\sqrt{n}$ 
6:      $M_q \leftarrow \text{MAX-MODULUS}(n, \lambda)$ 
7:      $m_q \leftarrow \text{MIN-MODULUS}(n, L, \text{scheme})$ 
8:     if  $m_q < M_q$  then
9:        $q \leftarrow m_q$ 
10:    else
11:       $n \leftarrow n + 1$ 
12:    end if
13:  until  $q \neq 0$ 
14:  return  $n, \sigma, q$ 
15: end function

```

5 Practical parameters

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

5.1 Multiplicative depth for an arbitrary binary circuit

Table 2 provides parameters for FV, SHIELD and F-NTRU for 80 and 128 bits of security. Parameters are extracted following the latest recommendations, that is to say $\sigma_{err} = 2\sqrt{n}$ for each scheme and $\sigma_{key} = 2n\sqrt{8nq} \cdot q^{1/3+e}$ for F-NTRU in order to maintain the security on the DSPR assumption [19].

First observation, F-NTRU seems less efficient. Even though the authors reported $n = 1024$ and $\log_2 q = 125$ bits for $L = 5$ and $\lambda = 80$ bits [8]. Due to equation 3, such a q is too high to maintain 80 bits of security for $n = 1024$, $\log_2 q$ should be less than 54 bits. This is why to find a q that enables both $L = 5$ and $\lambda = 80$ bits, the dimension should be much higher. Values for SHIELD seem the bests 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 $2 \times N = 4 \times \log_2 q = 472$ sub-polynomials of degree-2327 with 118 bits coefficients, whereas FV only requires two sub-polynomials of degree-3167 with 157 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 and F-NTRU are both 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

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	1012	87	1721	54	1295	87	2184
5	157	3167	191	3884	161	4218	195	5151
10	298	6082	333	6800	307	8190	341	9110
15	448	9138	482	9827	460	12311	494	13225
20	602	12246	636	12931	619	16573	653	17477

(b) Selection of parameters for SHIELD. (c) Selection of parameters for F-NTRU. Binary error, $\sigma_{key} = 2\sqrt{n}$. $\sigma_{key} = 2n\sqrt{8nq} \cdot q^{1/3+e}$ ($e = 2^{-64}$), $\sigma_{err} = 2\sqrt{n}$, $\omega = 16$ bits.

L	$\lambda = 80$ bits		$\lambda = 128$ bits		L	$\lambda = 80$ bits		$\lambda = 128$ bits	
	$\log_2 q$	n	$\log_2 q$	n		$\log_2 q$	n	$\log_2 q$	n
1	35	627	36	824	1	109	2161	111	2869
5	118	2327	121	3128	5	319	6510	323	8627
10	235	4792	240	6378	10	597	12147	605	16198
15	360	7343	367	9815	15	886	17930	898	24003
20	490	9989	500	13383	17	1004	20277	1017	27159

of multiplicative for binary messages. Table 3 provides parameters for SHIELD and F-NTRU for the optimized circuit. FV is omitted here, because it presents no particular optimization.

Results are very impressive, both schemes scale to large multiplicative depth 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 $n = 12246$ and $\log_2 q = 602$ bits for a multiplicative depth of 20. For the F-NTRU scheme, even with this optimization, parameters seems to high for a practical use. Indeed, the degree- n is just above 2048 for multiplicative depths from 1 to 20, implying a NTT NWC of size 4096 with coefficients larger than 100 bits (yet below 128 bits).

SHIELD is best for an optimized circuit. Since F-NTRU is far from competitive, we focus our study on FV and SHIELD in the next sections.

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

(a) Selection of parameters for SHIELD. (b) Selection of parameters for F-NTRU.
 Binary error, $\sigma_{key} = 2\sqrt{n}$. $\sigma_{key} = 2n\sqrt{8nq} \cdot q^{1/3+e}$, $\sigma_{err} = 2\sqrt{n}$.

L	$\lambda = 80$ bits		$\lambda = 128$ bits		L	$\lambda = 80$ bits		$\lambda = 128$ bits	
	$\log_2 q$	n	$\log_2 q$	n		$\log_2 q$	n	$\log_2 q$	n
1	35	627	36	824	1	109	2161	111	2869
5	38	693	39	899	5	113	2236	113	2974
10	39	715	40	923	10	115	2273	114	3026
15	40	736	41	947	15	116	2291	118	3052
20	40	736	41	947	20	116	2291	119	3077

5.3 The case of the Negative Wrapped Convolution

Attracted by its performance, a majority of polynomial multiplication implementation uses the NWC NTT. We provide in Table 4 the associated parameters for FV. For SHIELD, parameters seem quite independent of the multiplicative depth. Moreover, because the polynomial degree is oversized due to NWC, a security of $\lambda = 128$ bits is always achieved. For the same use case as FV, the polynomial degree is always 1024, with $\log_2 q = 39$ bits for a multiplicative depth of 5, 40 bits for a multiplicative depth of 10, and 41 bits for a multiplicative depth of 15. 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 $n = 2617$ and $\log_2 q = 130$. In a NWC NTT scenario, 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 3.4, the batching technique is very useful to reduce the ciphertext expansion. Table 5 provides parameters for FV and SHIELD when 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 3, the dimension goes over 1024 and implies an associated

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	54	1	×	×	×	×	×	×
2048	80	2	55	1	80	2	×	×
4096	186	6	161	5	135	4	141	3
8192	389	13	361	12	307	10	284	8
16384	793	26	764	25	589	19	591	18

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 292 sub-polynomials of degree 2949 with coefficients of 146 bits, while without batching it only requires 78 sub-polynomials of degree 715 with coefficients of 39 bits.

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 multiplicative depth of 15, it is 13.7 MB large, 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. Three different schemes have been studied: FV, a 2nd generation scheme, and two 3rd generation schemes: SHIELD and F-NTRU. As we showed, F-NTRU's parameters are really worse than SHIELD's ones, and then we do not recommend it for practical use. Hence, we focused on a more precise comparison of FV and SHIELD.

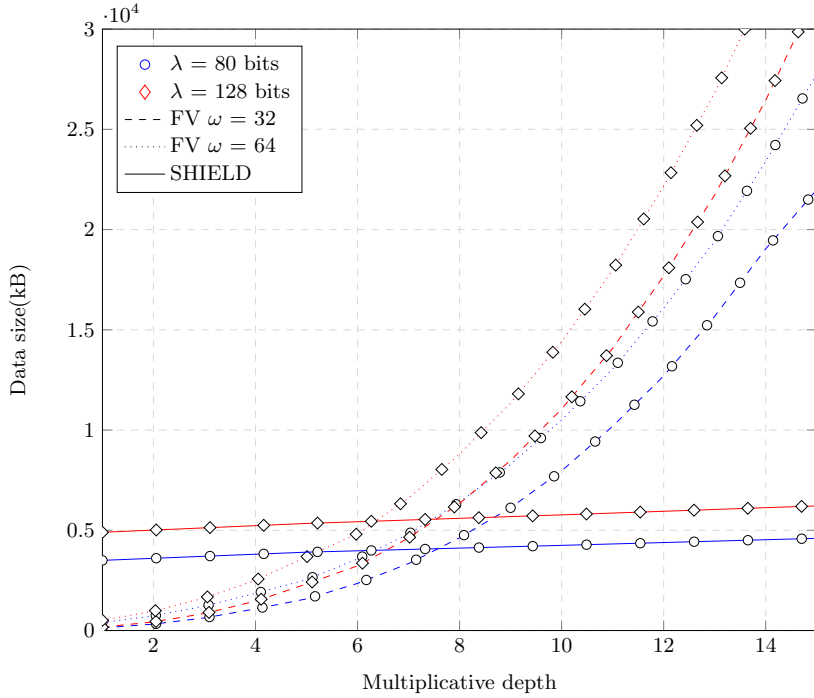


Fig. 1: Data size for an homomorphic scenario with 8 encryptions.

FV has in major cases shorter ciphertexts than the others, thanks to the re-linearization 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 3rd 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, $\log_2 q$ 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 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.

Concerning the batching. Unlike FV, SHIELD is very sensitive to batching. For a multiplicative depth of 4, SHIELD with batching requires $n = 1342$ and $\log_2 q = 70$. This has 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 worsen 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, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. Cryptology ePrint Archive, Report 2015/046, 2015.
3. 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.
4. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012*, pages 868–886. Springer, 2012.
5. 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.
6. 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.
7. Yuanmi Chen and Phong Q Nguyen. BKZ 2.0: Better lattice security estimates, 2013. http://www.di.ens.fr/~ychen/research/Full_BKZ.pdf.
8. Yarkın Doröz and Berk Sunar. Flattening ntru for evaluation key free homomorphic encryption. Cryptology ePrint Archive, Report 2016/315, 2016.
9. Junfeng Fan and Frederick Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012.
10. Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.
11. 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.
12. Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. Shield: Scalable homomorphic implementation of encrypted data-classifiers. *Accepted to IEEE Transactions on Computers*, 2015.
13. 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.
14. 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.

15. Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *Annual Cryptology Conference*, pages 138–154. Springer, 2010.
16. 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.
17. Chris Peikert. How (Not) to Instantiate Ring-LWE. Cryptology ePrint Archive, Report 2016/351, 2016.
18. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.
19. Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 27–47. Springer, 2011.

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.

(b) Values for SHIELD.

L	batching	hw	n	m	L	batching	hw	n	m
1	2	7	1080	2025	1	2	7	648	1215
	4	31	1200	1625		6	9	756	1323
	12	33	1296	2835		12	33	720	1575
	14	49	1176	1421		18	49	648	999
	20	57	1200	2475	2	2	7	1000	1875
	24	59	1440	2925		6	9	972	1701
2	2	7	2058	2401		10	17	1000	1375
	6	9	1764	3087		18	25	972	1539
	8	41	1600	2125	24	59	864	1755	
	18	49	1944	2997	3	2	17	1176	1715
20	57	2000	4125	4		31	1200	1625	
3	6	9	2268	3969		12	33	1296	2835
	10	17	2200	3025		14	49	1176	1421
	12	33	2160	4725	20	57	1200	2475	
	22	41	2420	2783	4	2	31	1368	1805
24	59	2592	5265	8		41	1600	2125	
4	2	7	3000	5625		24	59	1440	2925
	6	9	2916	5103	5	2	7	1800	3375
	18	25	2916	4617		6	9	1764	3087
5	2	7	3240	6075	6	2	7	2058	2401
	6	23	3528	4459		6	17	1944	3159
	12	33	3600	7875		18	49	1944	2997
	20	57	3600	7425	7	6	9	2268	3969
6	2	9	4116	7203		10	17	2200	3025
	12	33	3888	8505		12	33	2160	4725
	24	59	3744	7605		16	73	2176	4335
7	2	15	4860	8019	18	169	2268	3429	
	4	23	4896	7803					
	6	41	4860	7533					
	20	57	4400	9075					
	24	59	4320	8775					

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	13.3 KB	26.7 KB	36.6 KB	73.1 KB	17.1 KB	34.1 KB	46.4 KB	92.8 KB
5	121.4 KB	607 KB	181.1 KB	543.3 KB	165.8 KB	994.8 KB	245.2 KB	1 MB
10	442.5 KB	4.3 MB	552.8 KB	3.2 MB	613.9 KB	6 MB	758.4 KB	4.4 MB
15	999.5 KB	13.7 MB	1.1 MB	9 MB	1.4 MB	20.3 MB	1.6 MB	12.5 MB
20	1.8 MB	33.4 MB	2.0 MB	19.6 MB	2.4 MB	48.9 MB	2.7 MB	29.9 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	375 KB	5.4 KB	375 KB	5.4 KB	521.4 KB	7.2 KB	521.4 KB	7.2 KB
5	488.6 KB	6.4 KB	5.2 MB	32.3 KB	667.7 KB	8.6 KB	7.4 MB	44.3 KB
10	531 KB	6.8 KB	30 MB	105.1 KB	721.1 KB	9 KB	43 MB	145.9 KB
15	575 KB	7.2 KB	94.8 MB	226.8 KB	777.3 KB	9.5 KB	136.6 MB	316.5 KB
20	575 KB	7.2 KB	223 MB	402 KB	777.3 KB	9.5 KB	320 MB	559.2 KB