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

Vincent Migliore*, Guillaume Bonnoron†
*Université Bretagne Sud, vincent.migliore@univ-ubs.fr
†Chair of Naval Cyber Defense, guillaume.bonnoron@ecole-navale.fr

***Abstract*—Homomorphic encryption gets increasing attention lately, and for good reasons. Lots of the burdens from the initial proposals have been overcome and real applications become feasible. In this work, we propose a survey of the current best solutions together with a deep analysis of how to setup and size these schemes, for real. Our overall aim is to provide easy-to-use guidelines for implementation purposes.**

I. Introduction

Homomorphic Encryption schemes are a new promising tool in modern cryptography because they allow to carry out operations on enciphered data. Moreover, the mathematical objects used to construct HE schemes are immune to quantum attacks. The *Fully Homomorphic Encryption* (FHE) schemes, are so called because they allow two types of elementary operations, addition and multiplication, thus enabling to process virtually any algorithm over encrypted data. Figure 1 illustrates a basic client/server transaction in an homomorphic scenario.

Common cryptographic schemes sometimes present some homomorphic properties, e.g. for multiplication [RSA78] or addition [Pai99]. But it was only with the works from Alguilar [MGH10], Gentry [Gen09] that key ideas were introduced to support both types of operations, with limited restrictions. The first FHE schemes are based on hard lattice problems, with high complexity and poor flexibility. So to lighten the overhead of homomorphic capabilities, a more promising solution is *Somewhat Homomorphic Encryption* (SHE). By (upper-)bounding the number of homomorphic operations, SHE considerably reduces the size of ciphertext operands and associated costs. As of today many FHE/SHE schemes have been presented. They can be split into three families depending on the assumption serving as hardness ground: Approximate-Great Common Divisor (AGCD) based schemes [vDGHV10][CLT14], LTV based schemes [BLLN13], [DS16] and Ring-Learning With Errors (R-LWE) based schemes [BGV12], [Bra12], [FV12], [GSW13], [BV14], [KGV15].

For many years, FHE/SHE theoretical background has been moving. Thus, it has been a real challenge to draw practical parameters. In former publications, the authors usually present values for specific use-cases and do not address wide range of applications. Moreover these values should now be considered with caution. To address this main issue that stands in the way toward broader implementation and use of homomorphic encryption, we concisely and precisely present in this paper how the extraction of SHE parameters work. We make specific efforts to provide 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 decide to evaluate three most promising schemes: FV, SHIELD and F-NTRU. The main contributions of the paper are as follows:

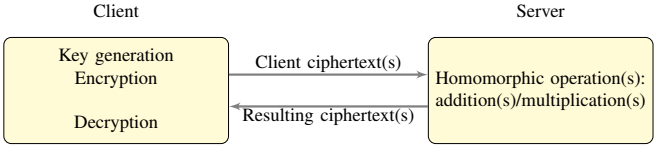

Fig. 1: Presentation of a client/server transaction in an homomorphic encryption scenario.

- A concise presentation of the three schemes with an harmonization of notation.
- A review of parameters extraction for each of them, with several explorations to evaluate parameters for different applications.
- Numerous tables of parameter under different constraints in order to cover a wide range of use cases.

This paper is organized as follows. Section II provides notation and the basic theoretical background required. Section III presents FV, SHIELD and F-NTRU with harmonized notation and provides a brief state of the art of current implementation techniques. Section IV provides methodology for parameters extraction. Section V proposes ready to use tables and compare FV, SHIELD and F-NTRU according to different scenarios. Section VI draws some conclusions.

II. Preliminaries

A. Notation

In the following, operations are performed on a ring of polynomials with integer coefficients. 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 $\Phi_m(X)$.

We adopt a particular notation for operands, described below. 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. Other standard operators are represented as follows:
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. These notations are 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:

- $l = \log_2 q$.
- $N = 2\ l$.
- $l_{\omega,q} = \lceil \log_2 q / \log_2 \omega \rceil$, for some integer ω

In the following, all polynomial operations are considered performed in R_q.

B. Ring-LWE

We recall here the definition of the Ring-Learning With Errors problem [LPR10].

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 $\sigma > 2\sqrt{n}$.

III. Presentation of the different SHE schemes

A. FV

FV [FV12] is a transposition of the scale-invariant Brakerski scheme [Bra12] to the Ring-LWE problem. The public key is a pair $(AS + E, A)$ of a Ring-LWE instance, and the secret key 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.
All primitives of FV are as follows:

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

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

- 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}) :
  $\mathbf{A} \leftarrow U_{R_q}^{l_{w,q}}$, $\mathbf{E} \leftarrow D_{R_q,\sigma_{err}}^{l_{w,q}}$
  $\gamma = \left(\left[\text{FV.PowersOf}_{w,q}\left(S_{key}^2\right) - \left(\mathbf{A}S_{key} + \mathbf{E}\right)\right]_q, \mathbf{A}\right)$
  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}) :
  $\widetilde{M} = \left[C[0] + C[1]S_{key}\right]_q$
  $m = \left\lfloor \frac{t}{q}\widetilde{M}[0] \right\rceil$
  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) :
  $\widetilde{C}_0 = \left[\left\lfloor \frac{t}{q} C_A[0] \times C_B[0] \right\rceil\right]_q$
  $\widetilde{C}_1 = \left[\left\lfloor \frac{t}{q} \left(C_A[0] \times C_B[1] + C_A[1] \times C_B[0]\right) \right\rceil\right]_q$
  $\widetilde{C}_2 = \left[\left\lfloor \frac{t}{q} C_A[1] \times C_B[1] \right\rceil\right]_q$
  $C_\times = \text{FV.Relin}(\widetilde{C}_0, \widetilde{C}_1, \widetilde{C}_2, \gamma)$
  return $C_\times$

- 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

B. SHIELD

SHIELD [KGV15] is a transposition of the GSW scheme [GSW13] to the Ring-LWE problem. It is a so called 3^{rd} generation FHE/SHE schemes, and does not require any relinearization. As an important issue, such schemes require 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 2^{nd} generation FHE/SHE schemes, reducing the size of the modulus q and the cyclotomic polynomial degree n. By carefully examining the construction of 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 FV.
All primitives of SHIELD are as follows:

- 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$(\mathbf{C}, \mathbf{S_{key}})$:
  $\mathbf{M} = \mathbf{C} \cdot \mathbf{S_{key}} = m \cdot \text{BDI}(\mathbf{I_{N\times N}}) \cdot \mathbf{S_{key}} + error$
  $m = \left\lfloor \frac{2}{q} \cdot \mathbf{M}[0][0] \right\rceil$
  return m
- SHIELD.Add$(\mathbf{C_1}, \mathbf{C_2})$:
  $\mathbf{C}_+ = \mathbf{C_1} + \mathbf{C_2}$
  return $\mathbf{C}_+$
- SHIELD.Mult$(\mathbf{C_1}, \mathbf{C_2})$:
  $\mathbf{C}_\times = \text{BD}(\mathbf{C_1}) \cdot \mathbf{C_2}$
  return $\mathbf{C}_\times$

C. F-NTRU

F-NTRU [DS16] scheme is the latest homomorphic encryption scheme presented in this paper. To understand how F-NTRU is constructed, it is important to understand the history of LTV-based schemes. In 2012, a strong competitor to FV scheme was introduced using one polynomial instead of two for the ciphertext, at a limited cost on the size of the modulus q. This scheme called YASHE' is based on the LTV scheme and introduces an additional constraint on the secret key, it must be an invertible polynomial. An additional assumption is required, the Decision Small Polynomial Ratio (DSPR), and has been shown to be insecure with the parameters previously selected. However, with immune parameters, the noise growth is too important to allow homomorphic operations. Then F-NTRU was introduced, using latest noise management techniques proposed in GSW to reduce the noise growth, and thus enabling homomorphic operations. This new noise management requires an additional operation called FLATTEN, requiring an array of polynomials instead of a single one like YASHE'.
All primitives of F-NTRU are as follows:

- F-NTRU.BDI$(\mathbf{A})$:
  $(\mathbf{A} \in B_{R_q}^{l\times l})$
  $\mathbf{B} \in R_q^{l\times 1}$
  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$(\mathbf{A})$:
  $(\mathbf{A} \in R_q^{l\times 1})$
  $\mathbf{B} \in B_{R_q}^{l\times l}$
  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$(\mathbf{A})$: $(\mathbf{A} \in B_{R_q}^{l\times l})$
  $\mathbf{B} \in B_{R_q}^{l\times l}$
  $B =$ F-NTRU.BD(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}) :
  $\mathbf{S} \leftarrow D^{l\times 1}_{R_q,\sigma_{err}}, \mathbf{E} \leftarrow D^{l\times 1}_{R_q,\sigma_{err}}$
  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
  $\mathbf{C}$ = F-NTRU.FLATTEN$(m \cdot \mathbf{I}_{l\times l} + \text{BD}(\mathbf{C'}_{l\times 1}))$

- F-NTRU.Decrypt(C, S_{key}) :
  $c_0 = \text{BDI}(C_{(0,l-1)}, \dots C_{(0,0)})$
  $m = \lfloor c_0 S_{key} \rceil mod 2$
  return m
- F-NTRU.Add$(\mathbf{C_1}, \mathbf{C_2})$:
  $\mathbf{C}_+$ = F-NTRU.FLATTEN$(\mathbf{C_1} + \mathbf{C_2})$
  return $\mathbf{C}_+$
- F-NTRU.Mult$(\mathbf{C_1}, \mathbf{C_2})$:
  $\mathbf{C}_\times$ = F-NTRU.FLATTEN$(\mathbf{C_1} \cdot \mathbf{C_2})$
  return $\mathbf{C}_\times$

D. Batching

For each scheme above, the cleartext is a polynomial in R_q. For convenience, messages are commonly chosen to be integers, in order to perform additions and multiplications on integers instead of polynomials. However, this integer representation turns out to be limited when considering interesting homomorphic operations. Indeed, constructing a simple conditional or test operator is impossible without a binary representation. In which case a ciphertext is an encryption of one bit of cleartext.
This latter representation brings two important issues. First, while the integer addition or multiplication requires one homomorphic operation in the case of integer representation of messages, the binary representation requires to reconstruct the binary circuit of these operators, which is clearly not efficient. Second, the size of the ciphertexts is strongly impacted.
To balance the ciphertext expansion issue, the batching technique is a good solution. Introduced in [SV14], the batching allows to "pack" several messages into one 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.

E. 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. That is why $x^n - 1$ and $x^n + 1$ are often chosen as they can be completely factorized with degree-1 factors. When performing a polynomial multiplication using the NTT algorithm, the output polynomial is reduced by the polynomial which 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, 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 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 III-D. Thus, the NWC is optimized for performance but is incompatible with bathing techniques.

IV. Parameters extraction

A. Methodology

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

B. Noise management

1) Notation: We briefly introduce additional notations for the noise extraction. For polynomials A and B, we define $\|A\|_\infty = \max_{0\le i<n} \mid a_i \mid$. 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:

$$\delta = sup\{\|A \cdot B\|_\infty / \|A\|_\infty \|B\|_\infty\} = n$$

2) FV: The noise bound has been thoroughly studied in [LN14], thus we only recall some key information below.
a) Initial noise: To determine the initial noise, we apply the decryption procedure on a fresh ciphertext. To simplify the operations, we consider an encryption of a 0.

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

Thus, the initial noise can be expressed as:

$$B_0 = B_{err}(1 + 2nB_{key}) \tag{1}$$

b) Multiplicative noise: Following the approach in [LN14], to ensure concreteness of FV, one must satisfy:

$$C_1^L B_0 + L C_1^{L-1} C_2 < (\Delta - r_t(q))/2 \tag{2}$$

where :
$C_1 = \delta t(4 + \delta B_{key})$
$C_2 = \delta^2 B_{key}(B_{key} + t^2) + \delta \omega l_{\omega,q} B_{err}$
$\Delta = \lfloor q/t \rfloor$
$r_t(q) = q - \Delta t$
For binary messages it yields:
$C_1 = n(4 + nB_{key})$
$C_2 = n^2 B_{key}(B_{key} + 1) + n\omega l_{\omega,q} B_{err}$
$\Delta = \lfloor q/2 \rfloor$
$r_t(q) = q - 2 \cdot \Delta$

3) SHIELD: The authors analyzed in detail the noise growth [KGV15], but they only provide an asymptotic evaluation. Below we develop the calculation in order to extract the constant terms. In this section, BD refers to SHIELD.BD and BDI to SHIELD.BDI.

a) Initial Noise: To determine the initial noise, we apply the decryption procedure on a fresh ciphertext. To simplify the operations, we consider an 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

$$\mathcal{E} = \mathbf{r_{N\times 1}}\cdot E + \mathbf{E_{N\times 2}}\cdot\mathbf{S_{key}}$$

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

Thus, the initial noise can be bounded by:

$$B_0 = B_{err}(1+n(1+B_{key})) \tag{3}$$

b) Multiplicative Noise: To determine the noise after an homomorphic multiplication in SHIELD, we apply the decryption procedure after the multiplication step. $\mathrm{MULT}(C,D) = \mathrm{BD}(C)\cdot D$

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

We set

$$\mathcal{E}_\times = m_2\cdot\mathcal{E}_1 + \mathrm{BD}(\mathbf{C_1})\cdot\mathcal{E}_2 \tag{4}$$

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 :

$$\|\mathcal{E}_\times[i]\|_\infty < m_2\cdot B_0^{(1)} + (2n\log_2 q)B_0^{(2)} < B_0(1+2n\log_2 q) \tag{5}$$

Then, by an immediate induction, the noise after L homomorphic multiplications can be expressed as:

$$B_L = B_0(1+2n\log_2 q)^L \tag{6}$$

To be able to decrypt without error after L homomorphic multiplications, the final noise must be lower than $q/2$:

$$q/2 > B_0(1+2n\log_2 q)^L \tag{7}$$

c) Better noise for multiplication: Unlike FV, noise in SHIELD grows slowly if a ciphertext is multiplied by a fresh one. By carefully examining equation 5, 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 :

$$B_L = B_0 + L(2n\log_2 q)B_0 = B_0(1+L(2n\log 2q)) \tag{8}$$

d) 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 can be expressed as:

$$B_{i+1} = n\cdot B_i + (2n\log_2 q)B_0 \tag{9}$$

It is an arithmetico-geometric sequence:

- $B_{i+1} = a\cdot B_i + b$
- $a = n$
- $b = 2n\log_2 qB_0$
- $r = \frac{b}{1-a}$
- $B_L = a^L(B_0 - r) + r$

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 refers to F-NTRU.BD, BDI to F-NTRU.BDI and FLATTEN to F-NTRU.FLATTEN.

a) Initial Noise: To determine the initial noise, we apply the decryption procedure on a fresh ciphertext. To simplify the operations, we consider an encryption of a 0.

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

Thus, the initial noise can be expressed as:

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

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

$$B_0 \le 2\,B_{err}(3\,nB_{key}+1) \tag{10}$$

b) Multiplicative noise: In F-NTRU, a ciphertext is a $l\times l$-matrix of degree-n binary polynomials. As proposed in [DS16], 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 F-NTRU.BD/F-NTRU.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:

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

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 [DS16] for more explanations):

$$c_j^{(i)} = \sum_{k=0}^{l_{w,q}-1} c_{(j,k)} \cdot \tilde{c}_k^{(i-1)} + c_j^{(i-1)}\tilde{m} + \tilde{c}_j^{(i-1)} m + 2^j m\tilde{m} \quad (11)$$

We set:

- $\|c_j^{(i)}\|_\infty = \|y_i\|_\infty$
- $\|c_{j,k}^{(i)}\|_\infty = \|y_T\|_\infty = \omega$

Then, the first row can be written:

$$y_i = l_{w,q} \cdot \tilde{y}_{i-1} \cdot y_T + y_{i-1}\tilde{m} + \tilde{y}_{i-1} m + m\tilde{m} \quad (12)$$

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 can be expressed as:

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

Thus, the noise can be expressed as:

$$B_{i+1} \leq (n \cdot l_{w,q} \cdot w + 2) B_i + 2B_{key} + 1 \quad (13)$$

It is an arithmetico-geometric sequence:

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

c) Better noise for multiplication: Like SHIELD, when a ciphertext is multiplied by a fresh one, the noise growth is lower. By considering equation 12 with $\|\tilde{y}_{i-1}\|_\infty = B_0$, the new noise growth can be expressed as:

$$B_i = l_{\omega,q} \cdot n \cdot B_0 \cdot \omega + B_{i-1} + B_0 + 2\, B_{key} + 1$$

$$B_L \leq L \cdot (B_0 \cdot (1 + l_{w,q} \cdot n \cdot \omega) + 2\, B_{key} + 1) + B_0 \quad (14)$$

C. Security

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. [APS15] 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 [Pei16].

TABLE I: 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
λ = 80 bits	54 bits	103 bits	201 bits	401 bits
λ = 128 bits	44 bits	81 bits	156 bits	307 bits

2) Ring-LWE: A common approach to determine the security parameters is to consider the advantage of the attacker at distinguishing Ring-LWE sample 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 random, hence the distinction objective. For more information, the reader can refer to [APS15, 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 [CN13] experimented with BKZ and provide time estimates to achieve root Hermite factors. So, following the work in [LN14], we get a minimal γ from a security objective. Then we get an upper bound on q:

$$\log_2 q \leq \min_{m>n} \frac{m^2 \ \log_2 \gamma(m, \lambda) + m \ log_2(\sigma/\alpha)}{m - n} \quad (15)$$

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

3) Determining security parameters: Real use-cases of FHE/SHE define requirements for the multiplicative depth L and a security level λ to achieve, then one needs to choose the corresponding security parameters.

a) 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 [LN14].

$\gamma(m) = a \cdot log_{10}(m) + b$

- For λ = 80 bits : a = 0.0005649115, b = 1.005907
- For λ = 128 bits : a = 0.0002924305, b = 1.005042

b) 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 15, one can determine an upper-bound of q.

c) 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.

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

```
function ChooseParam(scheme, L, λ)
    q ← 0
    n ← 1
    repeat
        σ ← 2√n
        M_q ← Max-modulus(n, λ)
        m_q ← Min-modulus(n, L, scheme)
        if m_q < M_q then
            q ← m_q
        else
            n ← n + 1
        end if
    until q ≠ 0
    return n, σ, q
end function
```

All the values that we report in the tables have been determined with Algorithm 1 and are no more optimistic than those from estimators in [APS15, Table 2].

V. Practical parameters

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

A. Multiplicative depth for an arbitrary binary circuit

Table II 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 [SS11].

First observation, F-NTRU seems less efficient. Even though the authors reported n = 1024 and $\log_2 q$ = 125 bits for L = 5 and λ = 80 bits [DS16]. Due to equation 15, 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 outperforms the other schemes.

TABLE II: 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	λ = 80 bits				λ = 128 bits			
	ω = 32 bits		ω = 64 bits		ω = 32 bits		ω = 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. Binary error, $\sigma_{key} = 2\sqrt{n}$.

L	λ = 80 bits		λ = 128 bits	
	$\log_2 q$	n	$\log_2 q$	n
1	35	627	36	824
5	118	2327	121	3128
10	235	4792	240	6378
15	360	7343	367	9815
20	490	9989	500	13383

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

L	λ = 80 bits		λ = 128 bits	
	$\log_2 q$	n	$\log_2 q$	n
1	109	2161	111	2869
5	319	6510	323	8627
10	597	12147	605	16198
15	886	17930	898	24003
17	1004	20277	1017	27159

B. 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 of multiplicative for binary messages. Table III 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 II, 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. Therefore we focus

TABLE III: Parameters for SHIELD and F-NTRU, where λ is the security level and L the multiplicative depth when the circuit is optimized as described in section V-B. Binary message (No batching).

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

L	λ = 80 bits		λ = 128 bits	
	$\log_2 q$	n	$\log_2 q$	n
1	35	627	36	824
5	38	693	39	899
10	39	715	40	923
15	40	736	41	947
20	40	742	41	952

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

L	λ = 80 bits		λ = 128 bits	
	$\log_2 q$	n	$\log_2 q$	n
1	109	2161	111	2869
5	113	2236	113	2974
10	115	2273	114	3026
15	116	2291	118	3052
20	116	2291	119	3077

TABLE IV: Parameters for FV and SHIELD 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.

(a) Parameters for FV.

n	λ = 80 bits				λ = 128 bits			
	ω = 32 bits		ω = 64 bits		ω = 32 bits		ω = 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

(b) Parameters for SHIELD.

n	λ = 80 bits		λ = 128 bits	
	$\log_2 q$	L	$\log_2 q$	L
	37	1	37	1
	39	5	39	5
1024	40	10	40	10
	41	15	41	15
	41	20	41	20

our study on FV and SHIELD in the next sections.

C. 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 IV specific parameters for FV and SHIELD in this setting. 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.

Also, as clearly visible on the table for SHIELD, the constraint on n tighten the choice in parameters and the usual parameter selection procedure yields the same value for both 80 and 128 bits of security for SHIELD.

D. The impact of batching

As stated in section III-D, the batching technique is very useful to reduce the ciphertext expansion. Tables V and VI provide parameters for respectively FV and SHIELD when batching technique is used, in an optimized circuit as described in section V-B. 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 to double the size of the associated NTT to 2048. Moreover, the modulus q grows significantly with the depth, on average 12 more bits per level. which lead 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.

E. Keys and ciphertexts sizes

Table VII provides sizes for keys and ciphertexts for FV and SHIELD in different scenarios. For FV, the relinearization key size is also provided because it is part of the required key material

We see that the size dependence in the security level is similar for both schemes, slightly worse for SHIELD.

For small multiplicative depths, namely under 10, FV and SHIELD have comparable ciphertexts size. But for larger depths, the better noise management in SHIELD is very beneficial when compared to FV. The size of the FV relinearization key also becomes very large with depth. 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 in batching cases.

TABLE V: Parameters of FV 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, $\sigma_{err} = 2\sqrt{n}$. ω = 32 bits.

L	batching	hw	n	m
1	2	7	1080	2025
	4	31	1200	1625
	12	33	1296	2835
	14	49	1176	1421
	20	57	1200	2475
	24	59	1440	2925
2	2	7	2058	2401
	6	9	1764	3087
	8	41	1600	2125
	18	49	1944	2997
	20	57	2000	4125
3	6	9	2268	3969
	10	17	2200	3025
	12	33	2160	4725
	22	41	2420	2783
	24	59	2592	5265
4	2	7	3000	5625
	6	9	2916	5103
	18	25	2916	4617
5	2	7	3240	6075
	6	23	3528	4459
	12	33	3600	7875
	20	57	3600	7425
6	2	9	4116	7203
	12	33	3888	8505
	24	59	3744	7605
7	2	15	4860	8019
	4	23	4896	7803
	6	41	4860	7533
	20	57	4400	9075
	24	59	4320	8775

Even a packing of only 3 accounts for the same as what we observe in FV with increase in the multiplicative depth.

VI. CONCLUSION

This study has provided practical information to use homomorphic encryption in practice. Three different schemes have been studied: a second generation scheme called FV, and two third generation schemes called SHIELD and F-NTRU.

FV has in major cases smaller dimensions than third generation schemes, thanks to the relinearization step. It is only composed of two polynomials, yet with higher degree and coefficient size than third generation schemes. Moreover, it is very sensitive to the multiplicative depth and has no particular optimization for a given binary circuit.

SHIELD is a third generation scheme, which means that the relinearization step is somehow included in the homomorphic multiplication. The noise growth is much better than second generation homomorphic encryption schemes, leading to ciphertexts composed of smaller sub-polynomials. Yet there are much more polynomials to handle, $\log_2 q$ times more. This is not 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.

On the batching side, unlike FV, SHIELD is very sensitive to its use. 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 now one requires 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 mul-

TABLE VI: Parameters of 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, $\sigma_{err} = 2\sqrt{n}$.

L	batching	hw	n	m
1	2	7	648	1215
	6	9	756	1323
	12	33	720	1575
	18	49	648	999
2	2	7	1000	1875
	6	9	972	1701
	10	17	1000	1375
	18	25	972	1539
	24	59	864	1755
3	2	17	1176	1715
	4	31	1200	1625
	12	33	1296	2835
	14	49	1176	1421
	20	57	1200	2475
4	2	31	1368	1805
	8	41	1600	2125
	24	59	1440	2925
5	2	7	1800	3375
	6	9	1764	3087
6	2	7	2058	2401
	6	17	1944	3159
	18	49	1944	2997
7	6	9	2268	3969
	10	17	2200	3025
	12	33	2160	4725
	16	73	2176	4335
	18	169	2268	3429

TABLE VII: Parameters size for FV and SHIELD, where λ is the security level and L the multiplicative depth. $\sigma_{key} = 1/9.2, \sigma_{err} = 2\sqrt{n}$. Optimized circuit for SHIELD.

(a) FV

L	λ = 80 bits			
	ω = 32 bits		ω = 64 bits	
	ciphertext Public key	Relin. key	ciphertext Public key	Relin. key
1	13.3 KB	26.7 KB	36.6 KB	73.1 KB
5	121.4 KB	607 KB	181.1 KB	543.3 KB
10	442.5 KB	4.3 MB	552.8 KB	3.2 MB
15	999.5 KB	13.7 MB	1.1 MB	9 MB
20	1.8 MB	33.4 MB	2.0 MB	19.6 MB

L	λ = 128 bits			
	ω = 32 bits		ω = 64 bits	
	ciphertext Public key	Relin. key	ciphertext Public key	Relin. key
1	17.1 KB	34.1 KB	46.4 KB	92.8 KB
5	165.8 KB	994.8 KB	245.2 KB	1 MB
10	613.9 KB	6 MB	758.4 KB	4.4 MB
15	1.4 MB	20.3 MB	1.6 MB	12.5 MB
20	2.4 MB	48.9 MB	2.7 MB	29.9 MB

(b) SHIELD

L	λ = 80 bits			
	No batching		Batching	
	ciphertext	Public key	ciphertext	Public key
1	375 KB	5.4 KB	375 KB	5.4 KB
5	488.6 KB	6.4 KB	5.2 MB	32.3 KB
10	531 KB	6.8 KB	30 MB	105.1 KB
15	575 KB	7.2 KB	94.8 MB	226.8 KB
20	575 KB	7.2 KB	223 MB	402 KB

L	λ = 128 bits			
	No batching		Batching	
	ciphertext	Public key	ciphertext	Public key
1	521.4 KB	7.2 KB	521.4 KB	7.2 KB
5	667.7 KB	8.6 KB	7.4 MB	44.3 KB
10	721.1 KB	9 KB	43 MB	145.9 KB
15	777.3 KB	9.5 KB	136.6 MB	316.5 KB
20	777.3 KB	9.5 KB	320 MB	559.2 KB

tiplicative is important, namely $L \geq 10$, and 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.

ACKNOWLEDGMENT

This study has been partially funded by the french Direction Générale de l'Armement (DGA).

REFERENCES

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. Cryptology ePrint Archive, Report 2015/046, 2015.

[BGV12] 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.

[BLLN13] 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.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012*, pages 868–886. Springer, 2012.

[BV14] 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.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *Proc. of Public-Key Cryptography – PKC 2014*, pages 311–328. Springer, 2014.

[CN13] Yuanmi Chen and Phong Q Nguyen. BKZ 2.0: Better lattice security estimates, 2013. http://www.di.ens.fr/~ychen/research/Full_BKZ.pdf.

[DS16] Yarkın Doröz and Berk Sunar. Flattening ntru for evaluation key free homomorphic encryption. Cryptology ePrint Archive, Report 2016/315, 2016.

[FV12] Junfeng Fan and Frederick Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[Gen09] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[GSW13] 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.

[KGV15] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. Shield: Scalable homomorphic implementation of encrypted data-classifiers. *Accepted to IEEE Transactions on Computers*, 2015.

[LN14] 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.

[LPR10] 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.

[MGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *Annual Cryptology Conference*, pages 138–154. Springer, 2010.

[Pai99] 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.

[Pei16] Chris Peikert. How (Not) to Instantiate Ring-LWE. Cryptology ePrint Archive, Report 2016/351, 2016.

[RSA78] Ronald Linn Rivest, Adi Shamir, and Leonard Max Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[SS11] 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.

[SV14] N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proc. of Advances in Cryptology – EUROCRYPT 2010*, pages 24–43. Springer, 2010.