Practical Parameters for *Somewhat* Homomorphic Encryption Schemes on Binary Circuits.

Vincent Migliore*, Guillaume Bonnoron†, Caroline Fontaine‡
*Université Bretagne Sud, Lorient, France, vincent.migliore@univ-ubs.fr
†Chair of Naval Cyber Defense, Lanvéoc, France, guillaume.bonnoron@ecole-navale.fr
‡CNRS/Lab-STICC UMR 6285& IMT Atlantique, Brest, France, caroline.fontaine@imt-atlantique.fr

Abstract—**Post-quantum cryptography gets increasing attention lately, as we have to prepare alternative cryptographic solutions that will resist attacks from quantum computers. A very large effort is being done to replace the usual primitives such as encryption, signature or authentication. This effort also pulls new cryptographic features such as Somewhat or Fully Homomorphic Encryption schemes, based on lattices. Since their introduction in 2009, lots of the burden has been overcome and real applications now become possible. However many papers suffer from the fast constant pace of evolution on the attack side, so their parameter analysis is usually incomplete or obsolete. In this work we present a thorough study of two schemes that have proven their worth: FV and SHIELD, providing a deep analysis of how to setup and size their parameters, to ensure both correctness and security. Our overall aim is to provide easy-to-use guidelines for implementation purposes.**

I. 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 [21] or addition [34]. But it was only with the works from [32] and [22] that key ideas were introduced to support both operations simultaneously. These schemes have been followed by many others [8], [18], [10], [9], [20], [23], [11], [26]. It is important to notice that nearly all these post-2009 schemes are built upon lattices, which introduces a great difference when comparing with former partial HE schemes, both on performance and security considerations. First the lattice-based homomorphic schemes are usually heavier in practice in terms of both efficiency and encrypted data size, yet they are quantum-resistant unlike the former partial HE schemes. 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 on their numbers, thus enabling to process virtually any algorithm over encrypted data. However, the first FHE schemes presented too many drawbacks for concrete use, 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, making them usable in practice. Some of these schemes allow to fix their parameters to reach a given upper bound we call the multiplicative depth. Among the many HE schemes that have been presented, the most promising ones are based on ideal lattices. Here, we focus on 2^{nd} and 3^{rd} generation schemes, which are the most efficient.

For many years, the theoretical background of homomorphic encryption schemes has been evolving. Thus, it has remained a real challenge to draw practical parameters. Moreover, former publications usually present values for specific use-cases and do not address a wide range of applications. This issue stands in the way toward broader implementation and use of homomorphic encryption, therefore to address this, we concisely and precisely present here how the extraction of SHE parameters works. Using the usual methods in the cryptography community, we make specific efforts to offer ready-to-use content to people from outside this community, providing pre-computed tables and simple formulas for a self-determination of parameters. The state-of-the-art in homomorphic encryption is in continuous evolution, both on efficiency and security front. The latest efficiency improvement comes with *bootstrapped* schemes [19], [15], [7] which remove the concerns about circuit multiplicative depth. Nevertheless, and despite their potentials, for concrete applications we need to consider sufficiently mature and tested solutions, to be sure that their security is strong enough. We recently saw examples of efficient schemes like Yashe[8] and F-NTRU[18] being damaged by the subfield/sublattice attack [1], [27]. Consequently, we consider that today the most practical SHE schemes in terms of both efficiency and security are: BGV[10], FV[20] and SHIELD[26]. Software implementations of the first two are available: for BGV there is HElib[24] and for FV there are SEAL [28] and FV-NFLlib [17]. These schemes are well-adapted to different settings as studied in [16], namely BGV is best for large plaintext moduli whilst FV and SHIELD are best for small plaintext moduli. We focus in this paper on binary plaintexts. Several works benefit from this setting, allowing to construct more generic operators such as comparison, and facilitating the articulation with non-homomorphic schemes to reduce the transmission overhead [33], [29], [12], [13]. A recent study [4] also supports this choice. This is why we provide detailed analysis of parameters settings ensuring good security levels for the two best SHE schemes fitting this kind of setting, FV and SHIELD.

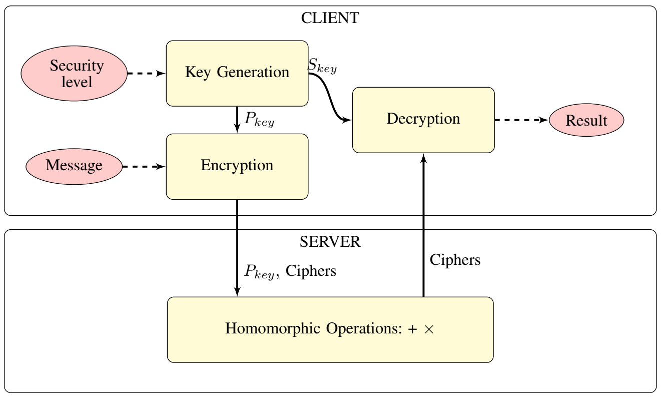


Fig. 1: Presentation of a client/server transaction in an homomorphic encryption scenario.

The main contributions of the paper are:

- a concise presentation of the two schemes with harmonized notation;
- a review of parameters extraction for FV and SHIELD, with several explorations to evaluate parameters for various applications, including cases never studied such as batching for SHIELD;
- numerous tables of parameters under different constraints, in order to cover a wide range of use cases.

One can easily compute more values with the method described here.
The paper is organized as follows. Section II provides notation and the basic theoretical background. Section III presents FV, SHIELD with harmonized notation and provides a brief state of the art of current implementation techniques. Section IV discusses the methodology for parameters extraction. Section V offers ready-to-use tables and compare these schemes according to different scenarios. Section VI draws some conclusions.

II. Preliminaries

A. Notation

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

B. Lattice introductory background

Lattices were first used in cryptology in the 1990s and serve today as one of the most promising ground for post-quantum schemes. In general, a lattice $\mathcal{L}$ of dimension n is a discrete additive subgroup of $\mathbb{R}^n$. *Integer* lattices are discrete additive subgroups of $\mathbb{Z}^n$. In this paper we only work with the integer lattices and simply call them lattices.

Lattices (of size n) are usually represented by a basis $\mathbf{B}$, a set of n independent integer vectors $(\mathbf{b_1}, \mathbf{b_2}, \dots, \mathbf{b_n})$ of size n whose integer linear combinations generate the lattice.

$$\mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{n} v_i \mathbf{b_i} : v_i \in \mathbb{Z} \right\} = \left\{ \mathbf{B}^T \mathbf{v} : \mathbf{v} \in \mathbb{Z}^n \right\} = \mathbf{B}\mathbb{Z}^n$$

In our lattices, $\mathbf{B}$ is always a square integer matrix, $\mathbf{B} \in \mathbb{Z}^{n \times n}$. For most of the discussion we restrict to this full rank definition and will make it explicit when working with greater generating families.

C. Ring-LWE

We recall here the definition of the Ring-Learning With Errors problem [31]. It was recently introduced as a Ring variant of the Learning With Errors problem from Regev [37] which is a very expressive hard problem that helped build many primitives, for the post-quantum era.
Definition *Let R be a ring of degree n over $\mathbb{Z}$ (usually $R = \mathbb{Z}[x]/(f(x))$ for some cyclotomic polynomial $f(x)$). Let q be a positive integer, χ a probability distribution on R of width parameter σ and S a secret random element in R_q. We denote by $L_{S,\chi}$ the probability distribution on $R_q \times R_q$ obtained by choosing $A \in R_q$ uniformly at random, choosing $E \in R$ according to χ and considering it in R_q, and returning $(A, C) = (A, [A \cdot S + E]_q) \in R_q \times R_q$.*

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

Search-Ring-LWE *is the problem to recovering* S *from pairs* (A, C) *sampled from* $L_{S,\chi}$.

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

III. Presentation of the schemes

A. FV

FV [20] is a transposition of the scale-invariant Brakerski scheme [9] 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 a 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. An integer t (much smaller than q) corresponds to the upper bound of a message. When $t = 2$, messages are binary. ω is a parameter associated with the relinearization, and determines the size of the relinearization key and the complexity of the relinearization operation. It is usual to select ω as a 32-bit or 64-bit integer for computational aspects.

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

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

```
• 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) :
  $\mathbf{A} \leftarrow U_{R_q}^{l_{w,q}}$, $\mathbf{E} \leftarrow D_{R_q,\sigma_{err}}^{l_{w,q}}$
  $\gamma = \left(\left[\text{FV.PowersOf}_{w,q}(S_{key}^2) - (\mathbf{A}S_{key} + \mathbf{E})\right]_q, \mathbf{A}\right)$
  return $\gamma$
• FV.Encrypt(m, P_key) :
  $U \leftarrow D_{R_q,\sigma_{key}}$ , $(E_1, E_2) \leftarrow D_{R_q,\sigma_{err}}^2$
  $C = \left(\left[\frac{q}{t}m + P_{key}[0]U + E_1\right]_q, \left[P_{key}[1]U + E_2\right]_q\right)$
  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) :
  $\widetilde{C}_0 = \left[\left\lfloor \frac{t}{q} C_A[0] \times C_B[0] \right\rceil\right]_q$
  $\widetilde{C}_1 = \left[\left\lfloor \frac{t}{q} (C_A[0] \times C_B[1] + C_A[1] \times C_B[0]) \right\rceil\right]_q$
  $\widetilde{C}_2 = \left[\left\lfloor \frac{t}{q} C_A[1] \times C_B[1] \right\rceil\right]_q$
  $C_\times$ = FV.Relin$(\widetilde{C}_0, \widetilde{C}_1, \widetilde{C}_2, \gamma)$
  return $C_\times$
• FV.Relin(C̃_0, C̃_1, C̃_2, γ) :
  $C_R = (C_{R,0}, C_{R,1})$
  $C_{R,0} = \left[\widetilde{C}_0 + \left\langle \text{FV.WordDecomp}_{w,q}(\widetilde{C}_2), \gamma[0] \right\rangle\right]_q$
  $C_{R,1} = \left[\widetilde{C}_1 + \left\langle \text{FV.WordDecomp}_{w,q}(\widetilde{C}_2), \gamma[1] \right\rangle\right]_q$
  return $C_R$
```

B. SHIELD

SHIELD [26] is a transposition of the GSW scheme [23] to the Ring-LWE problem. It is a so called 3rd 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 2nd generation HE schemes, reducing the size of the modulus q and the cyclotomic polynomial degree n. The plaintext modulus t can be chosen close to q, whereas with FV it must be small[1]. 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(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 $\mathbf{B}$
```

[1]For use with non-binary plaintexts, this can be an interesting feature.

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

return BGV.BitDecomp($C$)$^T \cdot \tau_{S_1\rightarrow S_2}$
```

C. 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 [39], the batching allows to "pack" several messages into one single ciphertext. To do so, the associated cyclotomic polynomial must be reducible in $\mathbb{Z}_2[X]$, and have only simple root factors. Then, a polynomial CRT is applied to pack the messages, with one message per factor. Recent research [4] demonstrates that using binary encodings for plaintext data is optimal.

D. Current implementation techniques and their limits

Since the chosen polynomial multiplication algorithm impacts the parameters, we briefly introduce standard polynomial multiplication algorithms. Four algorithms are usually implemented:

The schoolbook algorithm with an asymptotic complexity of $\mathcal{O}(n^2)$;
The Karatsuba-Ofman algorithm [25] with an asymptotic complexity of $\mathcal{O}(n^{1.58})$;
The Toom-Cook algorithm [40] with an asymptotic complexity of $\mathcal{O}(n^{1.465})$;
The FFT algorithm [36] with an asymptotic complexity of $\mathcal{O}(n \log n)$.

The final choice on the polynomial multiplication algorithm will greatly depend on the size of operands. As such, because SHE/FHE requires computation on million-bit operands, the polynomial multiplication is usually implemented with FFT. The key idea behind FFT is to evaluate a polynomial multiplication using polynomial interpolation. For polynomials with integer modular arithmetic, FFT can be adapted to such arithmetic by using the NTT algorithm. In practice, the NTT is a specialization of the FFT in a finite field, implying modular integer arithmetic with a prime modulus q. The formal definition of NTT is as follow: Lets N be a power of 2, q a prime modulus such as $q \equiv 1 \mod 2N$, and w be a primitive N^{th} root if unity in $\mathbb{Z}_q$. The NTT/iNTT of a given polynomial A is defined by:

$$\begin{aligned} \mathrm{NTT}(A)[i] &= \sum_{j=0}^{N-1} A[j]w^{ij} \\ \mathrm{iNTT}(A)[i] &= n^{-1} \sum_{j=0}^{N-1} A[j]w^{-ij} \end{aligned} \tag{1}$$

For two polynomials A and B, the polynomial multiplication can be computed as follow:

$$\mathrm{iNTT}(\mathrm{NTT}(A) \odot \mathrm{NTT}(B)) = A \cdot B \mod X^N - 1 \tag{2}$$

where $\odot$ denote the point-wise multiplication of 2 vectors. This NTT is also called *Positive Wrapped Convolution* (PWC) and is not perfectly suited for homomorphic encryption because $X^N - 1$ is not a cyclotomic polynomial. Thus, one must double the size of the NTT and then performing the polynomial modular reduction manually.
However, with a minor modification, the NTT can be tuned the *Negative Wrapped Convolution* (NWC) which computes $A \cdot B \mod X^N + 1$. Then, because $X^N + 1$ is a cyclotomic polynomial, the polynomial modular reduction is directly integrated during NTT computations. Let ψ be a $2N$ root of unity in $\mathbb{Z}_q$ such as $\phi^2 \equiv w \mod q$, $(\hat{A}, \hat{B}, \hat{C}) \in R_q$ such as $\hat{A}[i] = \phi^i A[i]$,$\hat{B}[i] = \phi^i B[i]$ and $C[i] = \phi^{-i}\hat{C}[i]$, then:

$$\begin{aligned} \hat{C} &= \mathrm{iNTT}(\mathrm{NTT}(\hat{A}) \odot \mathrm{NTT}(\hat{B})) \\ C &= A \cdot B \mod X^N + 1 \end{aligned} \tag{3}$$

For information, the best known algorithm to compute NTT is the Shönhage-Strassen algorithm [38], which computes the polynomial multiplication with a complexity of $\mathcal{O}(n \log n \log \log n)$.

However, the NWC has an important issue when dealing with batching. When factoring $X^n + 1$ in $\mathbb{Z}_2[X]$, 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-C. Thus, for binary messages, the NWC, which is optimized for performance, is not well suited for parallel computations. For non binary messages, it would be possible to find some configurations compatible with batching by selecting a particular modulus prime q.
The main issue of actual software implementations like implementations of BGV, FV in [24], [28] and [17] is the fact that only the NTT NWC is implemented (so no batching for binary messages). To enable batching, two approaches can be followed:
First, even if NTT NWC itself is not compatible with batching, the FFT algorithm can be adapted to various interpolation polynomials in order to compute the polynomial modular reduction during computations.
Second, the NTT can be used to compute polynomial multiplication (without polynomial modular reduction) followed by a polynomial modular reduction. As an example, in [5], authors of the paper have implemented FV with batching by using the NTT PWC for the polynomial multiplication part, and a Barrett [6] reduction for the polynomial modular reduction. The Barrett reduction itself has been implemented at a cost of roughly two polynomial multiplications.

IV. Parameters extraction

As described in Section III-C, 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.

A. Noise management

1) Notation: We briefly introduce additional notation 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 [30], 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:

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

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

Multiplicative noise. Following the approach in [30], to ensure concreteness of FV, we must have

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

where

$$\begin{aligned} C_1 &= \delta t(4 + \delta B_{key}) \\ C_2 &= \delta^2 B_{key}(B_{key} + t^2) + \delta \omega l_{\omega,q} B_{err} \\ \Delta &= \lfloor q/t \rfloor \\ r_t(q) &= q - \Delta t \end{aligned}$$

For binary messages it yields:

$$\begin{aligned} C_1 &= n(4 + nB_{key}) \\ C_2 &= n^2 B_{key}(B_{key} + 1) + n\omega l_{\omega,q} B_{err} \\ \Delta &= \lfloor q/2 \rfloor \\ r_t(q) &= q - 2 \cdot \Delta \end{aligned}$$

3) SHIELD: The authors of [26] only provided an asymptotic evaluation of SHIELD's noise growth. We develop below a more precise calculation, providing the constant terms. In this section, BD and BDI refer to SHIELD.BD and SHIELD.BDI respectively.
Initial noise. To determine the initial noise, we apply the decryption procedure on a fresh ciphertext, focusing on the encryption of a 0:

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

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

$$\begin{aligned} \|\mathcal{E}[i]\|_\infty &\le nB_{err} + B_{err} + n \cdot B_{err} \cdot B_{key} \\ &= B_{err}(1 + n(1 + B_{key})) \end{aligned}$$

Thus, the initial noise can be bounded by $B_0 = B_{err}(1 + n(1 + B_{key}))$.
Multiplicative noise. To determine the noise after a homomorphic multiplication in SHIELD, we apply the decryption procedure after the multiplication step. Recall that SHIELD.Mult$(\mathbf{C_1}, \mathbf{C_2}) = \mathrm{BD}(\mathbf{C_1}) \cdot \mathbf{C_2}$

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

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

$$\begin{aligned} \|\mathcal{E}_\times[i]\|_\infty &\le m_2 \cdot B_0^{(1)} + 2n \cdot \log_2 q \cdot B_0^{(2)} \\ &\le B_0(1 + 2n \cdot \log_2 q) \end{aligned} \quad (4)$$

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 4, one can deduce that the noise of each ciphertext is independent. Thus, the multiplicative noise growth can be more finely managed. When a ciphertext is multiplied by L other fresh ciphertexts, the noise growth can be expressed as $B_L = B_0 + L(2n\log_2 q)B_0 = B_0(1+L(2n\log 2q))$.

With batching. Earlier, we extracted noise parameters when $m \in \{0,1\}$. However, if one wants to use batch operations, the message is now a polynomial with coefficients in $\{0,1\}$, which has a significant impact on the noise growth in SHIELD. Lets express the new bound of the noise $\mathcal{E}_\times$:

$$\begin{aligned}\|\mathcal{E}_\times[i]\|_\infty &\le \|m_2\cdot\mathcal{E}_1\|_\infty + \|\mathrm{BD}(\mathbf{C_1})\cdot\mathcal{E}_2\|_\infty \\ &\le n\cdot\|m_2\|_\infty\|\mathcal{E}_1\|_\infty + 2n\cdot\log_2 q\cdot\|\mathcal{E}_2\|_\infty\end{aligned} \tag{5}$$

In the case of the optimized circuit for SHIELD, i.e. the second ciphertext is a fresh one, the noise new bound can be expressed as :

$$B_{i+1} = n\cdot B_i + 2n\cdot\log_2 q\cdot B_0 \tag{6}$$

It is an arithmetico-geometric sequence of the form

$$B_{i+1} = a\cdot B_i + b$$

where $a = n$ and $b = 2n\log_2 qB_0$. So $B_L = a^L(B_0 - r) + r$, with $r = \frac{b}{1-a}$.

B. Security

While the noise management determines the multiplicative depth and set a minimum q to ensure it, the security requirement upper-bound the size of the modulus for a given dimension n. This means that to ensure a given multiplicative depth, one must increase the dimension n to be able to increase the modulus q.

1) Attacks: As expected in cryptography, all the schemes presented here come with hardness results, provided by reductions to the Ring-LWE problem. This hard problem is one of the best candidates for post-quantum cryptography. There are no quantum attacks performing better the classical ones. Yet, beyond these asymptotic reductions, we need concrete hardness results to choose the scheme parameters according to a security level objective, e.g. 80 bits or 128 bits. Albrecht et al. [3] summarize the state-of-the-art of the attacks against LWE. All of them apply against ring instances which are particular cases. Another line of algebraic attacks exists also against Ring-LWE [35].

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 random, hence the distinction objective. For more information, the reader can refer to [3, Section 5.3].

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

n	2048	4096	8192	16384
λ = 80 bits	89 bits	174 bits	348 bits	695 bits
λ = 128 bits	59 bits	114 bits	224 bits	444 bits

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 to 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 [14] experimented with BKZ and provide time estimates to achieve root Hermite factors. So, following the work in [30], we get a minimal γ from a security objective. Then we get an upper bound on q

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

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.

2) LWE estimator: For convenience, it also possible to use Albrecht's estimator (available on BitBucket[2]) to estimate attacks performances. It turned out that the recent attack described in [2] was the best against our instances. For a brief overview we put in Table 1 the maximal allowed $\log q$ for several dimensions at 80 and 128 bits of security.

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.

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

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

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

V. Practical parameters

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

[2]`https://bitbucket.org/malb/lwe-estimator`, commit 61ac716

Algorithm 1 Determine (n, σ 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:         $M_q \leftarrow$ Max-modulus($n, \lambda$)
 7:         $m_q \leftarrow$ Min-modulus($n, L$, 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
```

TABLE II: Parameters for FV and SHIELD, 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	1188	87	1982	55	1878	88	3106
5	159	3711	193	4507	166	6014	200	7292
10	303	7120	337	7917	317	11625	351	12898
15	454	10715	489	11549	475	17507	509	18729
20	611	14405	645	15187	639	23491	673	24755

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

L	λ = 80 bits		λ = 128 bits	
	$\log_2 q$	n	$\log_2 q$	n
1	36	752	38	1247
5	120	2772	124	4454
10	238	5597	246	9005
15	364	8556	376	13834
20	495	11685	511	18838

A. Multiplicative depth for an arbitrary binary circuit

Table II provides parameters for FV and SHIELD for 80 and 128 bits of security. They are extracted in the proved-hardness regime, that is to say $\sigma_{err} = 2\sqrt{n}$ for each scheme
Values for SHIELD seem the best in the tables. However the number of sub-polynomials for a given ciphertext explodes because it is proportional to $\log_2 q$ for SHIELD. For example, with L = 5, a ciphertext in SHIELD contains $2 \times N = 4 \times \log_2 q = 480$ sub-polynomials of degree-2772 with 120 bits coefficients, whereas FV only requires two sub-polynomials of degree-3711 with 159 bits coefficients.
Consequently, in the case of an arbitrary binary circuit, FV is best.

B. Multiplicative depth for an optimized circuit

As stated in the previous section, SHIELD seems inefficient for arbitrary circuits. However, all third generation schemes 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 for this optimized circuit. FV is omitted here, because it presents no particular optimization.

TABLE III: Parameters for SHIELD, where λ is the security level and L the multiplicative degree. Optimized circuit. Binary message (No batching). Binary key, $\sigma_{err} = 2\sqrt{n}$.

L	λ = 80 bits		λ = 128 bits	
	$\log_2 q$	n	$\log_2 q$	n
1	36	752	38	1247
5	38	803	40	1323
10	40	849	41	1363
15	40	854	42	1396
20	41	869	43	1429

Results are very impressive, SHIELD scale to large multiplicative degree with nearly no impact on n and q. For SHIELD and for 80 bits of security, the modulus only increases by 5 bits between a multiplicative depth of 1 and 20 when the degree of the associated cyclotomic polynomial remains under 1024. As a reminder from Table II, FV requires at least $n = 14405$ and $\log_2 q = 611$ bits for a multiplicative depth of 20.

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

C. The case of the Negative Wrapped Convolution

Attracted by its performance, a majority of polynomial multiplication implementations use the NWC NTT. We provide in Table IV the associated parameters for FV. For SHIELD, parameters seem quite independent of the multiplicative depth. Because the polynomial degree is oversized due to NWC, a security of $\lambda = 80$ bits requires $n = 1024$, cf Table III, and we can then go to very high L. Similarly, $n = 2048$ is required for $\lambda = 128$ bits. For the same use case as FV for $\lambda = 80$ bits, the polynomial degree is always 1024, with $\log_2 q = 38$ bits for a multiplicative degree of 5, 40 bits for a multiplicative degree of 10, and 41 bits for a multiplicative depth of 20. As a reminder, NWC uses the cyclotomic polynomial $x^n + 1$ and the NTT computations are performed in the ring $\mathbb{Z}[x]/(x^n+1)$. Hence the polynomial reduction is directly integrated into NTT computations. This performance tweak comes at the cost of disabling the packing of several messages into one ciphertext, no batching possible. Parameters are selected to maximize the multiplicative depth for a given n, which is necessarily a power of 2, because the NWC NTT set the cyclotomic polynomial to $x^n + 1$. When compared to the previous case, this slightly increases the size of the modulus, for a given multiplicative depth. For example with FV, for a multiplicative depth of 4, optimized parameters are $n = 3065$ and $\log_2 q = 132$. 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.

TABLE IV: Parameters for FV and SHIELD in the case of the NWC NTT, where λ is the security level and L the multiplicative depth. Binary key, $\sigma_{err} = 2\sqrt{n}$. Reminder: 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
2048	79	2	87	1	55	1	×	×
4096	159	5	166	4	109	3	88	1
8192	333	11	337	10	195	6	200	5
16384	675	22	677	21	443	14	414	12

(b) Parameters for SHIELD.

λ = 80 bits, n=1024		λ = 128 bits, n=2048	
$\log_2 q$	L	$\log_2 q$	L
36	1	38	1
38	5	40	5
40	10	41	10
40	15	42	15
41	20	43	20

TABLE V: Parameters of SHIELD for 80 bits of security when batching is enabled, where λ is the security level and L the multiplicative depth. Binary key, $\sigma_{err} = 2\sqrt{n}$.

L	$\log_2 q$	n
1	36	752
2	47	1016
3	59	1306
4	71	1596
5	84	1911
10	149	3476

D. The case of batching in FV and SHIELD

As stated in Section III-C, the batching technique is very useful to reduce the ciphertext expansion. Table V provides parameters for SHIELD when the batching technique is used, in an optimized circuit as described in Section V-B. FV is not represented because batching does not modify security parameters. 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 NTT of size 2048. Moreover, the modulus q grows significantly with the depth, on average 12 more bits per level. which leads to more and more sub-polynomials for a given ciphertext. For a multiplicative depth of 10, SHIELD with batching requires 596 sub-polynomials of degree 3476 with coefficients of 149 bits, while without batching it only requires 160 sub-polynomials of degree 849 with coefficients of 40 bits. Because SHIELD is more practical for large multiplicative depth than FV, the use of batching with SHIELD is not recommended.

E. Focus on polynomials choice for batching with FV

We have investigated the structure and the repartition of cyclotomic polynomials in order to measure the practicality of batching. Because there is no known efficient NTT to perform polynomial modular reduction with arbitrary cyclotomic polynomial (apart from the cases of NWC with $X^n + 1$ as presented in Section V-C), the polynomial modular reduction must be implemented manually. Thus, batching can only be practical if we have cyclotomic polynomial that allow efficient reduction for various multiplicative depth.
Because the polynomial modular reduction complexity is closely related to the hamming weight of the cyclotomic polynomial (*i.e.* the number of non-zero monomials), we have minimized as much as possible this parameter. Table VI provides the conclusions of our investigation. We only have investigated batching for FV, since SHIELD suffers from several issues with batching discussed in Section V-D. For each multiplicative depth, we have extracted the four cyclotomic polynomials with the lowest hamming weight and compatible with batching. As can be seen, batching can be implemented for each multiplicative depth with various number of batches. In addition, for all parameters, the hamming weight is very small when compared to the degree of the cyclotomic polynomial, further supporting the practicality of batching.

F. Keys and ciphertexts sizes

One of the key aspect of Homomorphic Encryption is obviously the size of keys and ciphertexts. In order to fairly evaluate FV and SHIELD, we have compared the volume of data required for each scheme in a scenario requiring 8 bits of information. Figure 2 provides the conclusions of the study. 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. This is consequence that SHIELD requires large parameters for the first multiplicative depths. But for larger depths, the improved noise management of SHIELD is highly beneficial. The main issue for FV is the size of the relinearization key. For a multiplicative depth of 15, it is as large as 17.8 MB, when SHIELD does not require such a key. It can be reduced a bit by enlarging ω at an additional computation cost. In Fig. 3 and 4, we show for FV the respective sizes of the relinearization key and 8 ciphertexts. Both are transmitted from client to server, and we can see that going from $\omega = 32$ to $\omega = 64$ significantly decreases the relinearization key sizes, and then the transmission overhead on this client to server communication. It is interesting also to notice that when the server sends a computation result back to the client, it does not need to include this extra key material, and in this case only the orange bars corresponding to ciphertexts are of interest. Hence, these figures illustrate both the different transmission overhead for client-server and server-client communications, and the interest of using $\omega = 64$ instead of $\omega = 32$ for the client-server upload.

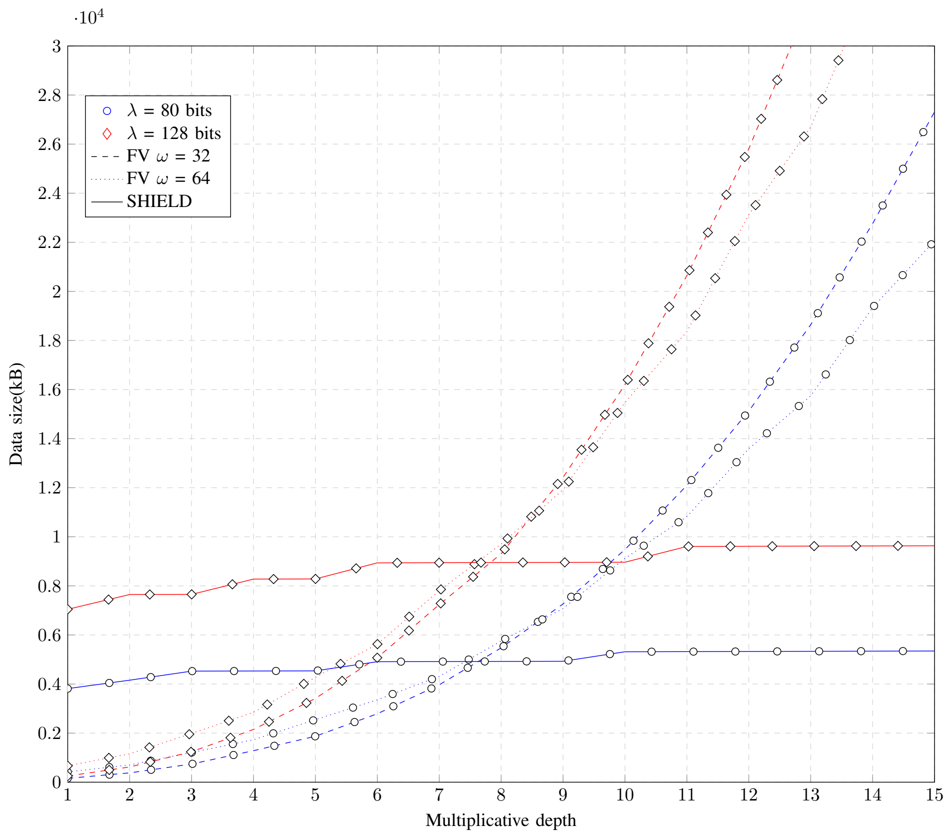
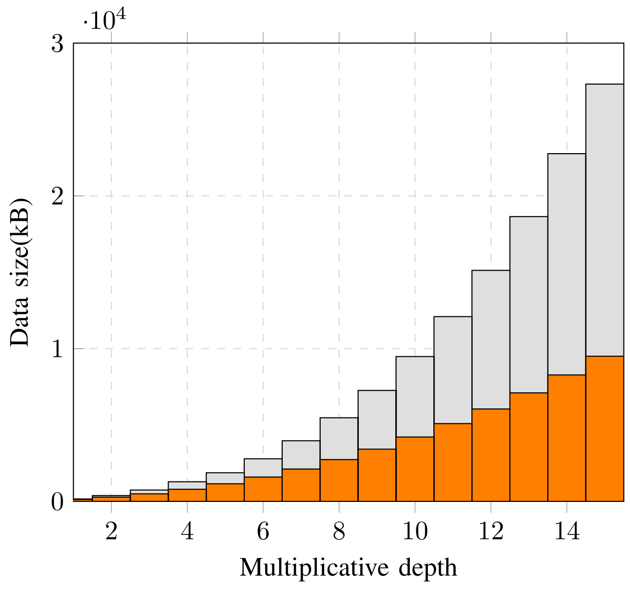
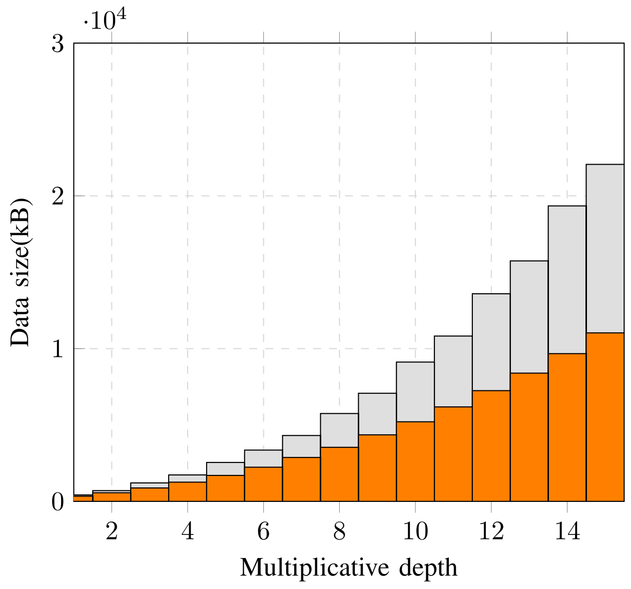


Fig. 2: Data size required for FV and SHIELD in an homomorphic scenario with 8 encryptions.



t

Fig. 3: Histograms showing the respective size of ciphertexts and keys for FV with $\lambda = 80$ and $\omega = 32$. The lower (orange) part is the cumulative size of 8 ciphertexts and the upper part is the relinearization key.



t

Fig. 4: Histograms showing the respective size of ciphertexts and keys for FV with $\lambda = 80$ and $\omega = 64$. The lower (orange) part is the cumulative size of 8 ciphertexts and the upper part is the relinearization key.

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

Range of n	L	batching	hw	m	Actual n
$[1024, 2048]$	1	2	7	3375	1800
		6	9	3087	1764
		12	33	2835	1296
		24	59	2925	1440
	2	2	7	3645	1944
		6	17	3159	1944
		18	49	2997	1944
		20	57	4125	2000
$[2048, 4096]$	3	2	7	5625	3000
		6	9	5103	2916
		18	25	4617	2916
		30	49	3875	3000
	4	2	7	6075	3240
		6	23	4459	3528
		12	33	7875	3600
		20	57	7425	3600
	5	2	17	6591	4056
		12	33	8505	3888
		24	59	7605	3744
		40	65	5125	4000
$[4096, 8192]$	6	2	7	9375	5000
		10	17	6875	5000
		12	33	11025	5040
		20	57	9075	4400
	7	2	7	10125	5400
		6	9	9261	5292
		18	25	9747	6156
		20	57	12375	6000

VI. Conclusion

This study has provided some new and helpful information concerning practical issues of homomorphic encryption for binary circuits. We have studied the two most practical schemes for this case: FV a second generation scheme, and SHIELD a third generation one. Our study covers parameter extraction for both, and explores the cases of NWC/NTT and batching. FV has in major cases shorter ciphertexts than SHIELD, thanks to the relinearization step. More precisely, an FV ciphertext is only composed of two polynomials, but with higher degree and coefficient size. However, FV is very sensitive to the multiplicative depth and has no particular optimization for any binary circuit. SHIELD is a third generation scheme, which means that the relinearization step is somehow included in the multiplication. The noise growth is much lower than for FV, leading to ciphertexts composed of smaller sub-polynomials. Yet there are many polynomials to handle, $2 \times \log_2 q$ times more. This is not a major issue for SHIELD because, if the computation is optimized to prefer multiplication with fresh ciphertexts, it can achieve a very high multiplicative depth (up to 20) without impacting much the sub-polynomial size. For example, maintaining it below 1024 for $\log_2 q \leq 41$ bits. As SHIELD authors reported, numerous but small polynomials multiplication can be very efficiently implemented in GPU and counterbalance the size of ciphertexts.

Concerning batching, SHIELD is, unlike FV, very sensitive to batching. For a multiplicative depth of 4, SHIELD with batching requires $n = 1596$ and $\log_2 q = 71$. This has a critical impact compared to the no-batching version because we now require to double the size of the NTT/NWC, and double the size of the integer multiplication operands. This phenomenon worsens when the multiplicative depth grows.
To conclude, SHIELD is a good candidate when the multiplicative depth is important, say $L \geq 10$, and even more when the computation involves fresh ciphertexts all along. But this only holds when the bandwidth is not 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 when coupled with the batching technique.

Future work on implementations could provide further insights on the real performances and behaviors of these schemes. Later when bootstrapped schemes will have gained maturity, they should be included in a similar analytical work. Also, for other kinds of applications requiring large plaintext moduli, it could be interesting to propose a similar study.

Acknowledgment

This study has been partially funded by the French Direction Générale de l'Armement (DGA), the French FUI's project CRYPTOCOMP and the French Chair of Naval Cyber Defense.

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. In *Proc. of CRYPTO*, 2016.
[2] Martin R Albrecht. On Dual Lattice Attacks Against Small-Secret LWE and Parameter Choices in HElib and SEAL. In *Proc. of EUROCRYPT*, 2017.
[3] Martin R. Albrecht, Rachel Player, and Sam Scott. On the Concrete Hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 2015.
[4] Angela Jäschke and Frederik Armknecht. (Finite) Field Work: Choosing the Best Encoding of Numbers for FHE Computation. In *Proc. of CANS*, 2017.
[5] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, Paulo Martins, Leonel Sousa, and Vincent Zucca. Efficient Reductions in Cyclotomic Rings - Application to Ring-LWE Based FHE Schemes. In *Proc. of SAC*, 2017.
[6] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Proc. of CRYPTO*, 1986.
[7] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from Tensored Homomorphic Accumulator. Cryptology ePrint Archive, Report 2017/996, 2017. http://eprint.iacr.org/2017/996.
[8] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig.
[9] Zvika Brakerski. Fully Homomorphic Encryption Without Modulus Switching from Classical GapSVP. In *Proc. of CRYPTO*, 2012.
[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proc. of ITCS*, 2012.
[11] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE As Secure As PKE. In *Proc. of ITCS*, 2014.
[12] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In *Proc. of FSE*, 2016.

[13] Sergiu Carpov, Thanh Hai Nguyen, Renaud Sirdey, Gianpiero Constantino, and Fabio Martinelli. Practical Privacy-Preserving Medical Diagnosis using Homomorphic Encryption. In *Proc. of Cloud Computing (CLOUD)*, 2016.

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

[15] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less than 0.1 Seconds. In *Proc. of ASIACRYPT*, 2016.

[16] Ana Costache and Nigel P Smart. Which Ring Based Somewhat Homomorphic Encryption Scheme is Best? In *Proc. of CT-RSA*, 2016.

[17] CryptoExperts. FV-NFLlib. Available at https://github.com/CryptoExperts/FV-NFLlib.

[18] Yarkın Doröz and Berk Sunar. Flattening NTRU for Evaluation Key Free Homomorphic Encryption. Cryptology ePrint Archive, Report 2016/315, 2016.

[19] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less than a Second. In *Proc. of EUROCRYPT*, 2015.

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

[21] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. Information Theory*, 1985.

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

[23] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Proc. of CRYPTO*, 2013.

[24] Shai Halevi. Helib. Available at https://github.com/shaih/HElib.

[25] A. Karatsuba and Y. Ofman. Multiplication of Multi-Digit Numbers on Automata (in Russian). *Doklady Akad. Nauk SSSR*, 1962.

[26] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Transactions on Computers*, 2015.

[27] Paul Kirchner and Pierre-Alain Fouque. Revisiting Lattice Attacks on Overstretched NTRU Parameters. In *Proc. of EUROCRYPT*, 2017.

[28] Kim Laine, Hao Chen, and Rachel Player. Simple Encrypted Arithmetic Library - SEAL (v2.3). Technical report, November 2017.

[29] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private Computation on Encrypted Genomic Data. In *Proc. of LATINCRYPT*, 2014.

[30] Tancrede Lepoint and Michael Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Proc. of AFRICACRYPT*, 2014.

[31] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning With Errors over Rings. *Proc. of EUROCRYPT*, 2010.

[32] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *Proc. of CRYPTO*, 2010.

[33] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can Homomorphic Encryption be Practical? In *Proc. of CCSW*, 2011.

[34] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. of EUROCRYPT*, 1999.

[35] Chris Peikert. How (Not) to Instantiate Ring-LWE. In *Proc. of SCN*, 2016.

[36] J.M. Pollard. The Fast Fourier Transform in a Finite Field. In *Mathematics of Computation*. 1971.

[37] Oded Regev. On Lattices, Learning With Errors, Random Linear Codes, and Cryptography. In *Proc. of STOC*, 2005.

[38] Arnold Schönhage and Volker Strassen. Fast Multiplication of Large Numbers. *Computing*, 1971.

[39] N. P. Smart and F. Vercauteren. Fully Homomorphic SIMD Operations. *Designs, Codes and Cryptography*, 2014.

[40] A. Toom. The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers. *Soviet Mathematics-Doklady*, 1963.

Vincent Migliore is a PhD student in Electrical and Computer Engineering with the Université Bretagne Sud, France. He received his M.Sc. degree from Ecole Normale Superieure de Rennes, France in 2014. His current research interests deal with the implementation of fast software/hardware computations for Fully/Somewhat Homomorphic Encryption.

Guillaume Bonnoron is a PhD student in Cryptography in the Chair of Naval Cyberdefense, in France. He received his M.Sc. degree from TELECOM Bretagne, France in 2013. His current research interests deal with closing the gap between theoretical proposals and real implementations of Somewhat/Fully homomorphic encryption.

Caroline Fontaine is a full-time researcher at CNRS (French National Research Institute). She has been working on content protection for more than 15 years. Her publications cover cryptography, steganography, digital watermarking, and active fingerprinting, with most of her articles tackling several of these domains at the same time. She has been and is involved in many research projects on these topics, and in the organization and program committees of many conferences and publications. She is with CNRS/Lab-STICC and Télécom Bretagne.