Which Ring Based Somewhat Homomorphic Encryption Scheme is Best?

Ana Costache and Nigel P. Smart

Dept. Computer Science,
University of Bristol,
Bristol, UK.
anamaria.costache@bristol.ac.uk,nigel@cs.bris.ac.uk

Abstract. The purpose of this paper is to compare side-by-side the NTRU and BGV schemes in their non-scale invariant (messages in the lower bits), and their scale invariant (message in the upper bits) forms. The scale invariant versions are often called the YASHE and FV schemes. As an additional optimization, we also investigate the ffect of modulus reduction on the scale-invariant schemes. We compare the schemes using the "average case" noise analysis presented by Gentry et al. In addition we unify notation and techniques so as to show commonalities between the schemes. We find that the BGV scheme appears to be more efficient for large plaintext moduli, whilst YASHE seems more efficient for small plaintext moduli (although the benefit is not as great as one would have expected).

1 Introduction

Some of the more spectacular advances in implementation improvements for Somewhat Homomorphic Encryption (SHE) schemes have come in the context of the ring based schemes such as BGV [3]. The main improvements here have come through the use of SIMD techniques (first introduced in the context of Gentry's original scheme [7] by Smart and Vercauteren [17], but then extended to the Ring-LWE based schemes by Gentry et al [3]). SIMD techniques in the ring setting allow for a small overall asymptotic overhead in using SHE schemes [8] by exploiting the Galois group to move data between slots. The Galois group can also be used to perform cheap exponentiation via the Frobenius endomorphism [9]. Other improvements in the ring based setting have come from the use of modulus switching to a larger modulus, so as to perform key switching [9], the use of scale invariant versions [6, 1], and the use of NTRU to enable key homomorphic schemes [14].

The scale invariant schemes, originally introduced in [2], are particularly interesting, they place the message space in the "upper bits" of the decryption equation, as opposed to the lower bits. This enables a more effective noise control mechanism to be employed which does not on the face of it require modulus switching to keep the noise within bounds. However, the downside is that they require a more complex rounding operation to be performed in the multiplication procedure.

However each paper which analyses the schemes uses a different methodology for deriving parameters, and examining the noise growth. In addition not all papers utilize all optimizations and improvements available. For example papers on the NTRU

scheme [5, 14], and its scale invariant version YASHE [1], rarely, if at all, make mention of the use of SIMD techniques. Papers working on scale invariant systems [6, 1] usually focus on plaintext moduli of two, and discount larger moduli. But many applications, e.g. usage in the SPDZ [4] MPC system, require the use of large moduli.

We have therefore conducted a systematic study of the main ring-based SHE schemes with a view to producing a fair comparison over a range of possible application spaces, from low characteristic plaintext spaces through to large characteristic ones, from low depth circuits through to high depth ones. The schemes we have studied are BGV, whose details can be found in [3, 8, 9], and its scale-invariant version [6] (called FV in what follows), the basic NTRU scheme [5, 14], and its scale-invariant version YASHE [1]. A previous study [12] only compared FV and YASHE, restricted to small plaintext spaces (in particular characteristic two), and did not consider the various variants in relation to key switching and modulus switching which we consider. Our results are broadly in line with [12] (where we have a direct comparison) for YASHE, but our estimates for FV appear slightly better.

On the face of it one expects that YASHE should be the most efficient, since it is scale invariant (which often leads to smaller parameters) and a ciphertext consists of only a single ring element, as opposed to two for the BGV style schemes. Yet this initial impression hides a number of details, wherein one can find a number of devils. It turns out that which is the most efficient scheme depends on the context (message characteristic and depth of admissible circuits).

To compare all four schemes fairly we apply the same API to all schemes, and the same optimizations. In particular we also investigate applying modulus switching to the scale invariant schemes (where its use is often discounted as not being needed). The use of modulus switching can be beneficial as it means ciphertexts become smaller as the function evaluation proceeds, resulting in increased performance. We also examine two forms of key switching (one based on the traditional decomposition technique and one based on raising the modulus to a larger value). For the decomposition technique we also examine the most efficient modulus to take in the modular decomposition, which turns out not to the two often seen in many treatments.

To compare the schemes we use the average distributional analysis first introduced in [9], which measures the noise in terms of the expected size in the canonical embedding norm. The use of the canonical embedding norm also deviates from some other treatments. For general rings the canonical embedding norm provides a more accurate measure of noise growth, over norms in the polynomial embedding, when analysed over a number of homomorphic operations. The noise growth of all of our schemes is analysed in the same way, and this is the first time (to our knowledge) that all schemes have been analysed on an equal footing.

The first question when performing such a comparison is how to compare security of differing schemes. On one hand one could take the standpoint of an exact security analysis and derive parameter sizes from the security theorems. However, even this is tricky when comparing schemes as the theorems may reduce security of different schemes to different hard problems. So instead we side-step this issue and select parameters according to an analysis of the best known attack on each scheme; which is luckily the same in all four cases. Thus we select parameters according to the Lindner-

Peikert analysis [13]. To also afford a fair comparison we use similar distributions for the various parameters for each scheme; e.g. small Hamming weight for the secret key distributions etc.

The next question is how to measure what is "better". In the context of a given specific scheme we consider one set of parameters to be better than another, for a given plaintext modulus, level bound and security parameter, if the number of bits to represent a ring element is minimized. After all this corresponds directly to the computational overhead when implementing the scheme. When comparing schemes one has to be a little more careful, as ciphertexts in the BGV family consist of two ring elements and in the NTRU family they consist of one element, but still ciphertext size is a good crude measure of overall performance. In addition, the operations needed for the scale invariant schemes are not directly compatible with the efficient double-CRT representation of ring elements introduced in [9], thus even if ciphertext sizes for the scale invariant schemes are smaller than for the non-scale invariant schemes, the actual computation times might be much larger.

As one can appreciate much of the analysis is an intricate following through of various inequalities. The full derivations can be found in the full version of this paper. We find that the BGV scheme appears to be more efficient for large plaintext moduli, whilst YASHE seems more efficient for small plaintext moduli (although the benefit is not as great as one would have expected).

2 Preliminaries

In this section we outline the basic mathematical background which forms the basis of our four ring-based SHE schemes. Much of what follows can be found in [8, 9], we recap on it here for convenience of the reader. We utilize rings defined by cyclotomic polynomials, $\mathbb{A} = \mathbb{Z}[X]/\Phi_m(X)$. We let $\mathbb{A}_q$ denote the set of elements of this ring reduced modulo various (possibly composite) moduli q. The ring $\mathbb{A}$ is the ring of integers of the mth cyclotomic number field $K = \mathbb{Q}(\zeta_m)$. We let $[a]_q$ for an element $a \in \mathbb{A}$ denote the reduction of a modulo q, with the set of representatives of coefficients lying in $(-q/2, \ldots, q/2]$, hence $[a]_q \in \mathbb{A}_q$. Assignment of variables will be denoted by $a \leftarrow b$, with equality being denoted by $=$ or $\equiv$.

Plaintext Slots: We will always use p for the plaintext modulus, and thus plaintexts will be elements of $\mathbb{A}_p$, and the polynomial $\Phi_m(X)$ factors modulo p into ℓ irreducible factors, $\Phi_m(X) = F_1(X) \cdot F_2(X) \cdots F_\ell(X) \pmod p$, all of degree $d = \phi(m)/\ell$. Just as in [3, 8, 17, 9] each factor corresponds to a "plaintext slot". That is, we view a polynomial $a \in \mathbb{A}_p$ as representing an ℓ-vector $(a \bmod F_i)_{i=1}^{\ell}$. We assume that p does not divide m so as to enable the slots to exist. In a number of applications p is likely to split completely in $\mathbb{A}$, i.e. $p \equiv 1 \pmod m$. This is especially true in applications not requiring bootstrapping, and hence only requiring evaluation of low depth arithmetic circuits.

Canonical Embedding Norm: Following the work in [15], we use as the "size" of a polynomial $a \in \mathbb{A}$ the l_∞ norm of its canonical embedding. Recall that the canonical

embedding of $a \in \mathbb{A}$ into $\mathbb{C}^{\phi(m)}$ is the $\phi(m)$-vector of complex numbers $\sigma(a) = (a(\zeta_m^i))_i$ where ζ_m is a complex primitive m-th root of unity and the indexes i range over all of $(\mathbb{Z}/m\mathbb{Z})^*$. We call the norm of $\sigma(a)$ the *canonical embedding norm* of a, and denote it by $\|a\|_\infty^{\mathsf{can}} = \|\sigma(a)\|_\infty$. We will make use of the following properties of $\|\cdot\|_\infty^{\mathsf{can}}$:

- For all $a, b \in \mathbb{A}$ we have $\|a \cdot b\|_\infty^{\mathsf{can}} \leq \|a\|_\infty^{\mathsf{can}} \cdot \|b\|_\infty^{\mathsf{can}}$.
- For all $a \in \mathbb{A}$ we have $\|a\|_\infty^{\mathsf{can}} \leq \|a\|_1$.
- There is a ring constant c_m (depending only on m) such that $\|a\|_\infty \leq c_m \cdot \|a\|_\infty^{\mathsf{can}}$ for all $a \in \mathbb{A}$.

where $\|a\|_\infty$ and $\|a\|_1$ refer to the relevant norms on the coefficient vectors of a in the power basis. The ring constant c_m is defined by $c_m = \|\mathsf{CRT}_m^{-1}\|_\infty$ where CRT_m is the CRT matrix for m, i.e. the Vandermonde matrix over the complex primitive m-th roots of unity. Asymptotically the value c_m can grow super-polynomially with m, but for the "small" values of m one would use in practice values of c_m can be evaluated directly. See [4] for a discussion of c_m.

Sampling From $\mathbb{A}_q$: At various points we will need to sample from $\mathbb{A}_q$ with different distributions, as described below. We denote choosing the element $a \in \mathbb{A}$ according to distribution $\mathcal{D}$ by $a \leftarrow \mathcal{D}$. The distributions below are described as over $\phi(m)$-vectors, but we always consider them as distributions over the ring $\mathbb{A}$, by identifying a polynomial $a \in \mathbb{A}$ with its coefficient vector.

The uniform distribution $\mathcal{U}_q$: This is just the uniform distribution over $(\mathbb{Z}/q\mathbb{Z})^{\phi(m)}$, which we identify with $(\mathbb{Z} \cap (-q/2, q/2])^{\phi(m)})$.

The "rounded Gaussian" $\mathcal{DG}_q(\sigma^2)$: Let $\mathcal{N}(0, \sigma^2)$ denote the normal (Gaussian) distribution on real numbers with zero-mean and variance σ^2, we use drawing from $\mathcal{N}(0, \sigma^2)$ and rounding to the nearest integer as an approximation to the discrete Gaussian distribution. The distribution $\mathcal{DG}_{q_t}(\sigma^2)$ draws a real ϕ-vector according to $\mathcal{N}(0, \sigma^2)^{\phi(m)}$, rounds it to the nearest integer vector, and outputs that integer vector reduced modulo q (into the interval $(-q/2, q/2]$).

Sampling small polynomials, $\mathcal{ZO}(p)$ and $\mathcal{HWT}(h)$: These distributions produce vectors in $\{0, \pm 1\}^{\phi(m)}$.

- For a real parameter $\rho \in [0, 1]$, $\mathcal{ZO}(p)$ draws each entry in the vector from $\{0, \pm 1\}$, with probability $\rho/2$ for each of -1 and $+1$, and probability of being zero $1 - \rho$.
- For an integer parameter $h \leq \phi(m)$, the distribution $\mathcal{HWT}(h)$ chooses a vector uniformly at random from $\{0, \pm 1\}^{\phi(m)}$, subject to the condition that it has exactly h nonzero entries.

Canonical embedding norm of random polynomials: In the coming sections we will need to bound the canonical embedding norm of polynomials that are produced by the

distributions above, as well as products of such polynomials. Following the work in [9] we use a heuristic approach, which we now recap on.

Let $a \in \mathbb{A}$ be a polynomial that was chosen by one of the distributions above, hence all the (nonzero) coefficients in a are independently identically distributed. For a complex primitive m-th root of unity ζ_m, the evaluation $a(\zeta_m)$ is the inner product between the coefficient vector of a and the fixed vector $\mathbf{z}_m = (1, \zeta_m, \zeta_m^2, \ldots)$, which has Euclidean norm exactly $\sqrt{\phi(m)}$. Hence the random variable $a(\zeta_m)$ has variance $V = \sigma^2\phi(m)$, where σ^2 is the variance of each coefficient of a. Specifically, when $a \leftarrow \mathcal{U}_q$ then each coefficient has variance $(q-1)^2/12 \approx q^2/12$, so we get variance $V_U = q^2 \cdot \phi(m)/12$. When $a \leftarrow \mathcal{DG}_q(\sigma^2)$ we get variance $V_G \approx \sigma^2 \cdot \phi(m)$, and when $a \leftarrow \mathcal{ZO}(\rho)$ we get variance $V_Z = \rho \cdot \phi(m)$. When choosing $a \leftarrow \mathcal{HWT}(h)$ we get a variance of $V_H = h$ (but not $\phi(m)$, since a has only h nonzero coefficients).

Moreover, the random variable $a(\zeta_m)$ is a sum of many independent identically distributed random variables, hence by the law of large numbers it is distributed similarly to a complex Gaussian random variable of the specified variance.[1] We therefore use $6\sqrt{V}$ (i.e. six standard deviations) as a high-probability bound on the size of $a(\zeta_m)$. Since the evaluation of a at all the roots of unity obeys the same bound, we use six standard deviations as our bound on the canonical embedding norm of a. (We chose six standard deviations since $\text{erfc}(6) \approx 2^{-55}$, which is good enough for us even when using the union bound and multiplying it by $\phi(m) \approx 2^{16}$.)

In this paper we model all canonical embedding norms as if from a random distribution. In [9] the messages were always given a norm of $\left\|m\right\|_\infty^{\mathsf{can}} \leq p \cdot \phi(m)/2$, i.e. a worst case bound. We shall assume that messages, and similar quantities, behave as if selected uniformly at random and hence estimate $\left\|m\right\|_\infty^{\mathsf{can}} \leq 6 \cdot p \cdot \sqrt{\phi(m)/12} = p \cdot \sqrt{3 \cdot \phi(m)}$. This makes our bounds better, and does not materially affect the decryption ability due to the larger effect of other terms. However, this simplification makes the formulae somewhat easier to parse.

In many cases we need to bound the canonical embedding norm of a product of two or more such "random polynomials". In this case our task is to bound the magnitude of the product of two random variables, both are distributed close to Gaussians, with variances σ_a^2, σ_b^2, respectively. For this case we use $16 \cdot \sigma_a \cdot \sigma_b$ as our bound, since $\text{erfc}(4) \approx 2^{-25}$, so the probability that both variables exceed their standard deviation by more than a factor of four is roughly 2^{-50}. For a product of three variables we use $40 \cdot \sigma_a \cdot \sigma_b \cdot \sigma_c$, since $\text{erfc}(3.4) \approx 2^{-19}$, and $3.4^3 \approx 40$.

3 Ring Based SHE Schemes

We refer to our four schemes as BGV, FV, NTRU and YASHE. The various schemes have been used/defined in various papers: for example one can find BGV in [3, 8, 9], FV in [6], NTRU in [5, 14] and YASHE in [1]. In all four schemes we shall use a chain of moduli for our homomorphic evaluation[2] by choosing L "small primes"

[1] The mean of $a(\zeta_m)$ is zero, since the coefficients of a are chosen from a zero-mean distribution.

[2] This is not strictly needed for the Scale invariant version if modulus switching is not performed.

$p_0, p_1, \ldots, p_{L-1}$ and the t^{th} modulus in our chain is defined as $q_t = \prod_{j=0}^{t} p_j$. A chain of L primes allows us to perform $L-1$ multiplications. The primes p_i's are chosen so that for all i, $\mathbb{Z}/p_i\mathbb{Z}$ contains a primitive m-th root of unity, i.e. $p_i \equiv 1 \pmod{m}$. Hence we can use the double-CRT representation, see [9], for all $\mathbb{A}_{q_t}$.

For the BGV and NTRU schemes we additionally assume that $p_i \equiv 1 \pmod{p}$. This is to enable the Scaling operation to work without having to additionally scale by $p_i \pmod{p}$, which would result in slightly more noise growth. A disadvantage of this is that the moduli p_i will need to be slightly larger than would otherwise be the case. The two scale invariant schemes (FV and YASHE) will make use of a scaling factor Δ_q defined by $\Delta_q = \left\lfloor \frac{q}{p} \right\rfloor = \frac{q}{p} - \epsilon_q$, where $0 \leq \epsilon_q < 1$.

3.1 Key Generation

We utilize the following methods for key generation, they sample the secret key in all cases, from a sparse distribution, this follows the choices made in [9]. This leads to more efficient homomorphic operations (since noise growth depends on the size of the secret key in many situations). However, such choices might lead to security weaknesses, which would need to be considered in any commercial deployment.

$\underline{\mathsf{KeyGen}^{\mathsf{BGV}}()}$: Sample $\mathfrak{sk} \leftarrow \mathcal{HWT}(h)$, $a \leftarrow \mathcal{U}_{q_{L-1}}$, and $e \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$. Then set the secret key as $\mathfrak{sk}$ and the public key as $\mathfrak{pk} \leftarrow (a, b)$ where $b \leftarrow [a \cdot \mathfrak{sk} + p \cdot e]_{q_{L-1}}$.

$\underline{\mathsf{KeyGen}^{\mathsf{FV}}()}$: Sample $\mathfrak{sk} \leftarrow \mathcal{HWT}(h)$, $a \leftarrow \mathcal{U}_{q_{L-1}}$, and $e \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$. Then set the secret key as $\mathfrak{sk}$ and the public key as $\mathfrak{pk} \leftarrow (a, b)$ where $b \leftarrow [a \cdot \mathfrak{sk} + e]_{q_{L-1}}$.

$\underline{\mathsf{KeyGen}^{\mathsf{NTRU}}()}$: Sample $f, g \leftarrow \mathcal{HWT}(h)$. Then set the secret key as $\mathfrak{sk} \leftarrow p \cdot f + 1$ and the public key as $\mathfrak{pk} \leftarrow [p \cdot g / \mathfrak{sk}]_{q_{L-1}}$. Note, if $p \cdot f + 1$ is not invertible in $\mathbb{A}_{q_{L-1}}$ we repeat the sampling again until it is.

$\underline{\mathsf{KeyGen}^{\mathsf{YASHE}}()}$: Sample $f, g \leftarrow \mathcal{HWT}(h)$. Then set the secret key as $\mathfrak{sk} \leftarrow p \cdot f + 1$ and the public key as $\mathfrak{pk} \leftarrow [p \cdot g / \mathfrak{sk}]_{q_{L-1}}$. Again, if $p \cdot f + 1$ is not invertible in $\mathbb{A}_{q_{L-1}}$ we repeat the sampling until it is.

3.2 Encryption and Decryption

The encryption algorithms for all four schemes are given in Fig. 1. As for key generation we select slightly simpler distributions than the theory would imply so as to ensure noise growth is not as bad as it would otherwise be. The output of each algorithm is a tuple $\mathfrak{c}$ consisting of the ciphertext data, the current level, plus a bound on the current "noise" B^*_{clean}. This bound is on the canonical embedding norm of a particular critical quantity which comes up in the decryption process; a different critical quantity depending on which scheme we are using. If the critical quantity has canonical embedding norm less than a specific value then decryption will work, otherwise decryption will likely fail. Thus having each ciphertext carry around an upper bound on the norm of this quantity allows us to analyse noise growth dynamically.

$\underline{\mathsf{Enc}^{\mathsf{BGV}}_{\mathfrak{pk}}(m)}$:
- $v \leftarrow \mathcal{ZO}(0.5)$.
- $e_0, e_1 \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$.
- $c_0 \leftarrow [b \cdot v + p \cdot e_0 + m]_{q_{L-1}}$,
- $c_1 \leftarrow [a \cdot v + p \cdot e_1]_{q_{L-1}}$,
- Output $\mathfrak{c} \leftarrow (c_0, c_1, L-1, B^{\mathsf{BGV}}_{\mathsf{clean}})$.

$\underline{\mathsf{Enc}^{\mathsf{FV}}_{\mathfrak{pk}}(m)}$:
- $v \leftarrow \mathcal{ZO}(0.5)$.
- $e_0, e_1 \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$.
- $c_0 \leftarrow [b \cdot v + e_0 + \Delta_{q_{L-1}} \cdot m]_{q_{L-1}}$,
- $c_1 \leftarrow [a \cdot v + e_1]_{q_{L-1}}$,
- Output $\mathfrak{c} \leftarrow (c_0, c_1, L-1, B^{\mathsf{FV}}_{\mathsf{clean}})$.

$\underline{\mathsf{Enc}^{\mathsf{NTRU}}_{\mathfrak{pk}}(m)}$:
- $e_0, e_1 \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$.
- $c \leftarrow [e_1 \cdot \mathfrak{pk} + p \cdot e_0 + m]_{q_{L-1}}$,
- Output $\mathfrak{c} \leftarrow (c, L-1, B^{\mathsf{NTRU}}_{\mathsf{clean}})$.

$\underline{\mathsf{Enc}^{\mathsf{YASHE}}_{\mathfrak{pk}}(m)}$:
- $e_0, e_1 \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$.
- $c \leftarrow [e_1 \cdot \mathfrak{pk} + e_0 + \Delta_{q_{L-1}} \cdot m]_{q_{L-1}}$,
- Output $\mathfrak{c} \leftarrow (c, L-1, B^{\mathsf{YASHE}}_{\mathsf{clean}})$.

Fig. 1: Encryption Algorithms for BGV, FV, NTRU and YASHE

To understand the critical quantity we have to first look at the decryption procedure in each case. Then we can apply our heuristic noise analysis to obtain an upper bound on the canonical embedding norm of the critical quantity for a fresh ciphertext, and so obtain B^*_{clean}; a process which is done in the full version of this paper.

$\underline{\mathsf{Dec}^{\mathsf{BGV}}_{\mathfrak{pk}}(\mathfrak{c})}$: Decryption of a ciphertext (c_0, c_1, t, ν) at level t is performed by setting $m' \leftarrow [c_0 - \mathfrak{sk} \cdot c_1]_{q_t}$, and outputting $m' \bmod p$. If we define the critical quantity to be $c_0 - \mathfrak{sk} \cdot c_1 \pmod{q_t}$, then this procedure will work when ν is an upper bound on the canonical embedding norm of this quantity and $c_m \cdot \nu < q_t/2$. If ν satisfies this inequality then the value of $c_0 - \mathfrak{sk} \cdot c_1 \pmod{q_t}$ will be produced exactly with no wrap-around, and will hence be equal to $m + p \cdot v$, if $c_0 = \mathfrak{sk} \cdot c_1 + p \cdot v + m \pmod{q_t}$. Thus we must pick the smallest prime $q_0 = p_0$ large enough to ensure that this always holds.

$\underline{\mathsf{Dec}^{\mathsf{FV}}_{\mathfrak{pk}}(\mathfrak{c})}$: Decryption of a ciphertext (c_0, c_1, t, ν) at level t is performed by setting

$$m' \leftarrow \left\lceil \frac{p}{q_t} \cdot [c_0 - \mathfrak{sk} \cdot c_1]_{q_t} \right\rfloor,$$

and outputting $m' \bmod p$. Consider the value of $[c_0 - \mathfrak{sk} \cdot c_1]_{q_t}$ computed during decryption, suppose this is equal to (over the integers before reduction mod q_t) $m \cdot \Delta_{q_t} + w + r \cdot q_t$. Then another way of looking at decryption is that we perform rounding on the value

$$\begin{aligned} \frac{p \cdot \Delta_{q_t} \cdot m}{q_t} + \frac{p \cdot w}{q_t} + \frac{p \cdot r \cdot q_t}{q_t} &= \frac{p \cdot (\frac{q_t}{p} - \epsilon_{q_t}) \cdot m}{q_t} + \frac{p \cdot w}{q_t} + p \cdot r \\ &= m + p \cdot \frac{w - \epsilon_{q_t} \cdot m}{q_t} + p \cdot r \end{aligned}$$

and then take the result modulo p. Thus the critical quantity in this case is the value of $w - \epsilon_{q_t} \cdot m$. So that the rounding is correct we require that ν is an upper bound on

$\left\|w - \epsilon_{q_t} \cdot m\right\|_\infty^{\mathsf{can}}$. The decryption procedure will then work when $c_m \cdot \nu < \Delta_{q_t}/2$, since in this case we have

$$\left\|p \cdot \frac{w - \epsilon_{q_t} \cdot m}{q_t}\right\|_\infty \leq \frac{c_m \cdot p}{q_t} \cdot \left\|w - \epsilon_{q_t} \cdot m\right\|_\infty^{\mathsf{can}} \leq \frac{\Delta_{q_t} \cdot p}{2 \cdot q_t} < \frac{1}{2}.$$

Thus again we must pick the smallest prime $q_0 = p_0$ large enough, to ensure that $c_m \cdot \nu < \Delta_{q_t}/2$.

$\underline{\mathsf{Dec}_{\mathfrak{pk}}^{\mathsf{NTRU}}(\mathfrak{c})}$: Decryption of a ciphertext (c, t, ν) at level t is performed by setting $m' \leftarrow [c \cdot \mathfrak{sk}]_{q_t}$, and outputting $m' \bmod p$. Much as with BGV the critical quantity is $[c \cdot \mathfrak{sk}]_{q_t}$. If ν is an upper bound on the canonical embedding norm of $c \cdot \mathfrak{sk}$, and we have $c = a \cdot \mathfrak{pk} + p \cdot e + m$ modulo q_t, for some values of a and e, then over the integers we have

$$[c \cdot \mathfrak{sk}]_{q_t} = m + p \cdot (a \cdot g + e + f \cdot m) + p^2 \cdot e \cdot f,$$

which will decrypt to m. Thus for decryption to work we require that $c_m \cdot \nu < q_t/2$.

$\underline{\mathsf{Dec}_{\mathfrak{pk}}^{\mathsf{YASHE}}(\mathfrak{c})}$: Decryption of a ciphertext (c, t, ν) at level t is performed by setting

$$m' \leftarrow \left\lceil \frac{p}{q_t} \cdot [c \cdot \mathfrak{sk}]_{q_t} \right\rfloor,$$

and outputting $m' \bmod p$. Following the same reasoning as for the FV scheme, suppose $c \cdot \mathfrak{sk}$ is equal to (again over the integers before reduction mod q_t) $m \cdot \Delta_{q_t} + w + r \cdot q_t$. Then for decryption to work we require ν to be an upper bound on $\left\|w - \epsilon_{q_t} \cdot m\right\|_\infty^{\mathsf{can}}$ and $c_m \cdot \nu < q_t/2$.

3.3 Scale

These operations scale a ciphertext, reducing the corresponding level and more importantly scaling the noise. The syntax is $\mathsf{Scale}^*(\mathfrak{c}, t_{out})$ where $\mathfrak{c}$ is at level t_{in} and the output ciphertext is at level t_{out} with $t_{out} \leq t_{in}$. The noise is scaled by a factor of approximately $q_{t_{in}}/q_{t_{out}}$, however an additive term of B_{scale}^* is added. For each of our variants see the full version of this paper for a justification of the proposed method and an estimate on B_{scale}^*.

For use in one of the $\mathsf{SwitchKey}^*$ variants we also use a Scale which takes a ciphertext with respect to modulus Q and produces a ciphertext with respect to modulus q, where $q|Q$. The syntax for this is $\mathsf{Scale}^*(\mathfrak{c}, Q)$; the idea here is that Q is a "temporary" modulus unrelated to the actual level t of the ciphertext, and we aim to reduce Q down to q_t. The former scale function can be defined in terms of the latter via

$\underline{\mathsf{Scale}^*(\mathfrak{c}, t_{out})}$:

- Write $\mathfrak{c} = (c, t, \nu)$.
- $\mathfrak{c}' \leftarrow \mathsf{Scale}^*((c, t_{out}, \nu), q_t)$.
- Output $\mathfrak{c}'$.

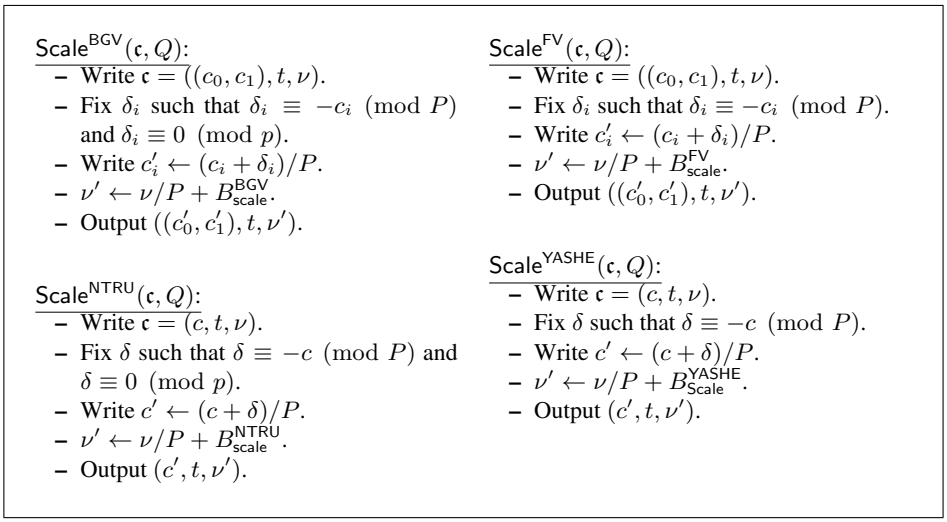

Fig. 2: Scale Algorithms for BGV, FV, NTRU and YASHE. In all methods $Q = q_t \cdot P$, and for the BGV and NTRU schemes we assume that $P \equiv 1 \pmod p$.

The Scale^* function was originally presented in [3] as a form of noise control for the non-scale invariant schemes. However, the use of such a function within the scale invariant schemes can also provide more efficient schemes, as alluded to in [6]. This is due to the modulus one is working with which decreases as homomorphic operations are applied. It is also needed for our second key switching variant. We thus present a Scale^* function for all our four schemes in Fig. 2.

3.4 Reduce Level

For all schemes we can define a $\mathsf{ReduceLevel}^*$ operation which reduces a ciphertext level from level t' to level t where $t' \geq t$. For the non-scale invariant schemes when we reduce a level we only perform a scaling (which could be an expensive operation) if the noise is above some global bound B. This is because for small noise we can easily reduce the level by just dropping terms off the modulus, since the modulus is a product of primes. For the scale invariant schemes we actually need to perform a Scale operation since we need to modify the Δ_{q_t} term. See the full version of this paper for details. In our parameter estimation evaluation we examine the case, for FV and YASHE, of applying modulus switching to reduce levels and not applying it. In the case of not applying it all ciphertexts remain at level $L - 1$, and $\mathsf{ReduceLevel}^*$ becomes a NOP.

3.5 Switch Key

The switch key operation is needed to relinearize after a multiplication, or after the application of a Galois automorphism (see [8] for more details on the latter). For all schemes we present two switch key operations:

– One based on decomposition modulo a general modulus T. See [11] for this method explained in the case of the BGV scheme. Unlike prior work we do not take $T = 2$, as we treat T as a parameter to be optimized to achieve the most efficient scheme. Although to ease parameter search we restrict to T being a power of two.
– Our second method is based on the raising the modulus idea from [9], where it was applied to the BGV scheme. Here we adopt a more complex switching operation, and a potentially larger parameter set, but we gain by reducing the size of the switching "matrices".

For each variant we require algorithms SwitchKeyGen and SwitchKey; the first generates the public switching "matrix", whilst the second performs the actual switch key. In the BGV and FV schemes we perform a general key switch of the underlying decryption equation of the form $d_0 - \mathfrak{sk} \cdot d_1 + \mathfrak{sk}' \cdot d_2 \longrightarrow c_0 - \mathfrak{sk} \cdot c_1$. For the NTRU and YASHE schemes the underlying key switch is of the form $c \cdot \mathfrak{sk}' \longrightarrow c' \cdot \mathfrak{sk}$. In Fig. 3 we present the key switching methods for the BGV algorithm. See the full version of this paper for the methods for the other schemes, plus derivations of upper bounds on the constants $B_{\mathsf{Ks},*} * (*)$.

$\underline{\mathsf{SwitchKeyGen}_1^{\mathsf{BGV}}(\mathfrak{sk}', \mathfrak{sk}, T)}$:
- For $i = 0$ to $\left\lceil \log_T(q_{L-1}) \right\rceil - 1$ do
  - $a_i \leftarrow \mathcal{U}_{q_{L-1}}$.
  - $e_i \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$.
  - $b_i \leftarrow [a_i \cdot \mathfrak{sk} + p \cdot e_i + T^i \cdot \mathfrak{sk}']_{q_{L-1}}$.
- $\mathfrak{ksk} \leftarrow (T, \{a_i, b_i\}_{i=0}^{\lceil \log_T q_{L-1} \rceil - 1})$.
- Output $\mathfrak{ksk}$.

$\underline{\mathsf{SwitchKey}_1^{\mathsf{BGV}}(\mathfrak{ksk}, (\mathfrak{d}, t, \nu))}$:
- Write d_2 in base T as $d_2 = \sum_{i=0}^{\lceil \log_T q_t \rceil - 1} d_{2,i} \cdot T^i$.
- $c_0 \leftarrow d_0 + \sum_{i=0}^{\lceil \log_T q_t \rceil - 1} d_{2,i} \cdot b_i \pmod{q_t}$.
- $c_1 \leftarrow d_1 + \sum_{i=0}^{\lceil \log_T q_t \rceil - 1} d_{2,i} \cdot a_i \pmod{q_t}$.
- $\nu' \leftarrow \nu + B_{\mathsf{Ks},1}^{\mathsf{BGV}}(t)$.
- Output $((c_0, c_1), t, \nu')$.

$\underline{\mathsf{SwitchKeyGen}_2^{\mathsf{BGV}}(\mathfrak{sk}', \mathfrak{sk})}$:
- $a \leftarrow \mathcal{U}_{q_{L-1}}$.
- $e \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$.
- $b \leftarrow [a \cdot \mathfrak{sk} + p \cdot e + P \cdot \mathfrak{sk}']_{q_{L-1} \cdot P}$.
- $\mathfrak{ksk} \leftarrow\leftarrow (a, b)$.
- Output $\mathfrak{ksk}$.

$\underline{\mathsf{SwitchKey}_2^{\mathsf{BGV}}(\mathfrak{ksk}, (\mathfrak{d}, t, \nu))}$:
- $c_0 \leftarrow [P \cdot d_0 + b \cdot d_2]_{q_t \cdot P}$.
- $c_1 \leftarrow [P \cdot d_1 + a \cdot d_2]_{q_t \cdot P}$.
- $\nu' \leftarrow P \cdot \nu + B_{\mathsf{Ks},2}^{\mathsf{BGV}}(t)$.
- Output $\mathsf{Scale}^{\mathsf{BGV}}(((c_0, c_1), t, \nu'), q_t \cdot P)$.

Fig. 3: The two variants of Key Switching for BGV.

In the context of BGV the first method requires us to store $\log_T(q_{L-1})$ "encryptions" of $\mathfrak{sk}'$, each of which is an element in $R^2_{q_{L-1}}$. The second method requires us to store a single "encryption" of $P \cdot \mathfrak{sk}'$, but this time as an element in $R^2_{P \cdot q_{L-1}}$. The former will require more space than the latter as soon as $\log_2 P < \log_T(q_{L-1})$. In terms of noise the output noise of the first method is modified by an additive constant of

$$B_{\mathsf{Ks},1}^{\mathsf{BGV}}(t) = \frac{8}{\sqrt{3}} \cdot p \cdot \left\lceil \log_T q_t \right\rceil \cdot \sigma \cdot \phi(m) \cdot T.$$

whilst the output noise of the second method is modified by the additive constant

$$\frac{B^{\mathsf{BGV}}_{\mathsf{Ks},2}(t)}{P} + B^*_{\mathsf{scale}} = \frac{8 \cdot p \cdot q_t \cdot \sigma \cdot \phi(m)}{\sqrt{3} \cdot P} + B^*_{\mathsf{scale}}.$$

As the level decreases this becomes closer and closer to B^*_{scale}, as the P in the denominator will wipe out the numerator term. Thus the noise will grow of the order of $O(\sqrt{\phi(m)})$ using the second method and as $O(\phi(m))$ using the first method. A similar outcomes arises when comparing the two methods with respect to the other three schemes.

3.6 Addition and Multiplication

We can now turn to presenting the homomorphic addition and multiplication operations. For reasons of space we give the addition and multiplication methods in the full version of this paper. In all methods the input ciphertexts $\mathbf{c}_i$ have level t_i, and recall our parameters are such that we can evaluate circuits with multiplicative depth $L-1$.

3.7 Security and Parameters

In this section we outline how we select parameters in the case where $\mathsf{ReduceLevel}^*$ is not a NOP (a no-operation). An analysis, for the FV and YASHE schemes, where $\mathsf{ReduceLevel}^*$ is a NOP we defer the analysis to the full version of this paper. We let B denote an upper bound on ν at the output of any $\mathsf{ReduceLevel}^*$ operation. Following [9] we set $B = 2 \cdot B^*_{\mathsf{scale}}$. We assume that operations are performed as follows. We encrypt, perform up to ζ additions, then do a multiplication, then do ζ additions, then do a multiplication and so on, where we assume decryption occurs after a multiplication.

Security: We assume, as a heuristic assumption, that if we set the parameters of the ring and modulus as per the BGV scheme then the other schemes will also be secure. We follow the analysis in [9], which itself follows on from the analysis by Lindner and Peikert [13][3]. We therefore have one of two possible lower bounds for $\phi(m)$, for security parameter k

$$\phi(m) \geq \begin{cases} \frac{\log(q_{L-1}/\sigma)\cdot(k+110)}{7.2} & \text{If the first variant of } \mathsf{SwitchKey} \text{ is used,} \\ \\ \frac{\log(P\cdot q_{L-1}/\sigma)\cdot(k+110)}{7.2} & \text{If the second variant of } \mathsf{SwitchKey} \text{ is used.} \end{cases} \tag{1}$$

Note the logs here are natural logarithms.

Bottom Modulus: To ensure decryption correctness at level zero we require that

$$4 \cdot c_m \cdot B^*_{\mathsf{scale}} = 2 \cdot c_m \cdot B < \begin{cases} p_0 & \text{For BGV and NTRU} \\ \\ \left\lfloor \frac{p_0}{p} \right\rfloor & \text{For FV and YASHE.} \end{cases} \tag{2}$$

[3] One could take into account a more elaborate analysis here, for example looking at BKW style attacks e.g. [10]. But for simplicity we follow the same analysis as in [9].

Top Modulus: At the top level we take as input a ciphertext with noise B^*_{clean}, perform ζ additions to produce a ciphertext with noise $B_1 = \zeta \cdot B^*_{\mathsf{clean}}$. We then perform a multiplication to produce something with noise

$$B_2 = \begin{cases} F^*(B_1, B_1) + B^*_{\mathsf{Ks},1}(L-1) & \text{If the first variant of } \mathsf{SwitchKey} \text{ is used,} \\ F^*(B_1, B_1) + \frac{B^*_{\mathsf{Ks},2}(L-1)}{P} + B^*_{\mathsf{scale}} & \text{If the second variant of } \mathsf{SwitchKey} \text{ is used.} \end{cases}$$

We then scale down a level to obtain something at the next level down. Thus we obtain something with noise bounded by $B_3 = \frac{B_2}{p_{L-1}} + B^*_{\mathsf{scale}}$. We require, for our invariant, $B_3 \leq B = 2 \cdot B^*_{\mathsf{scale}}$. Thus we require,

$$p_{L-1} \geq \frac{B_2}{B^*_{\mathsf{scale}}}. \tag{3}$$

Middle Moduli: A similar argument applies for the middle moduli, but now we start off with a ciphertext with bound $B = 2 \cdot B^*_{\mathsf{scale}}$ as opposed to B^*_{clean}. Thus we form

$$B'(t) = \begin{cases} F^*(\zeta \cdot B, \zeta \cdot B) + B^*_{\mathsf{Ks},1}(t) & \text{First variant of } \mathsf{SwitchKey}, \\ F^*(\zeta \cdot B, \zeta \cdot B) + \frac{B^*_{\mathsf{Ks},2}(t)}{P} + B^*_{\mathsf{scale}} & \text{Second variant of } \mathsf{SwitchKey}. \end{cases}$$

after which a Scale operation is performed. Hence, the modulus p_t for $t \neq 0, L-1$ needs to be selected so that

$$p_t \geq \frac{B'(t)}{B^*_{\mathsf{scale}}}. \tag{4}$$

Note, in practice we can do a bit better in the second variant of $\mathsf{SwitchKey}$ by merging the final two final scalings into one.

Putting It All Together: We are looking for parameters which satisfy equations (1), (2), (3) and (4), and which also minimize the size of data being processed, which is

$$\phi(m) \cdot \left(\sum_{t=0}^{L-1} p_t \right).$$

To do this we iterate through all possible values of $\log_2 q_{L-1}$ and $\log_2 T$ (resp. $\log_2 P$). We then determine $\phi(m)$, as the smallest value which satisfies equation (1). Here, we might need to take a larger value than the right hand side of equation (1) due to application requirements on p or the amount of packing required.

We then determine the size of p_{L-1} from equation (3), via

$$p_{L-1} \approx \left\lceil \frac{B_2}{B^*_{\mathsf{scale}}} \right\rceil.$$

We can now iterate downwards for $t = L-2, \ldots, 1$ by determining the size of $\log_2 q_t$, via

$$\log_2 q_t = \log_2 q_{t+1} - \log_2 p_{t+1}.$$

If we obtain $\log_2 q_t < 0$ then we abort, and pass to the next pair of $(\log_2 q_{L-1}, T)$ (resp. $(\log_2 q_{L-1}, \log_2 P)$) values. The value of p_t being determined by equation (4), via

$$p_t \approx \left\lceil \frac{B'(t)}{B^*_{\mathsf{scale}}} \right\rceil.$$

Finally we check whether a prime p_0 the size of $\log_2 q_0$, will satisify equation (2), if so we accept this set of values as a valid set of parameters, otherwise we pass to the next pair of $(\log_2 q_{L-1}, T)$ (resp. $(\log_2 q_{L-1}, \log_2 P)$) values.

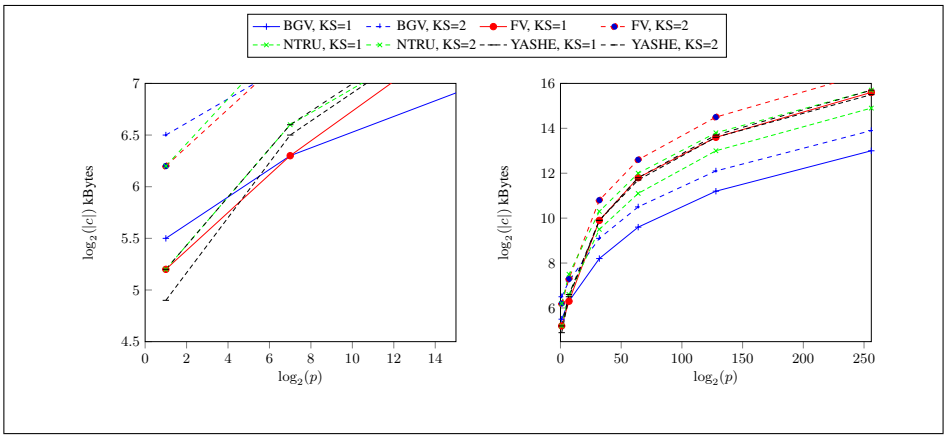

Fig. 4: Size of required ciphertext for various sizes of plaintext modulus when $L = 5$. The graph on the left zooms into the portion of the right graph for small values of $\log_2 p$.

4 Results

In the full version of this paper one can find a full set of parameters for each scheme, and variant of key switching, for various values of the plaintext modulus p and the number of levels L. In this section we summarize the overall conclusion. As a measure of efficiency we examine the size of a ciphertext in kBytes; this is a very crude measure but it will capture both the size of any data needed to be transmitted as well as the computational cost of dealing with a single ciphertext element within a calculation. In the full version of this paper we also examine the size of the associated key switching matrices, which is significantly smaller for the case of our second key switching method. In a given application this additional cost of holding key switching data may impact on the overall choices, but for this section we ignore this fact.

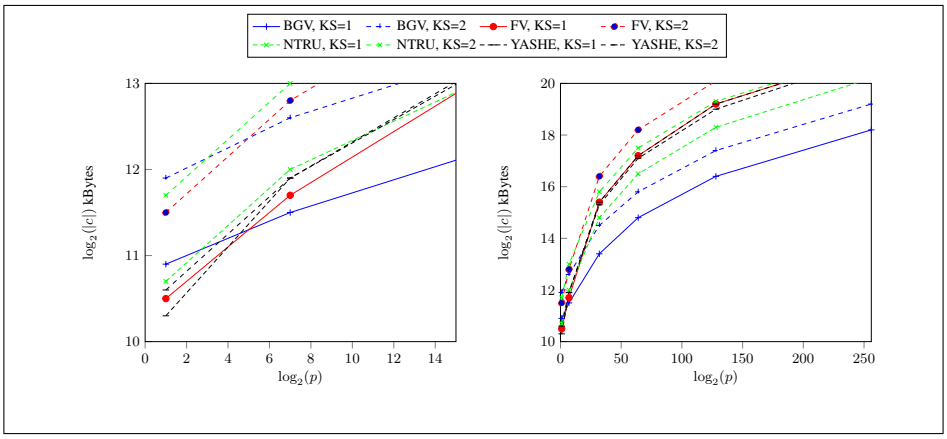

Fig. 5: Size of required ciphertext for various sizes of plaintext modulus when $L = 30$. The graph on the left zooms into the portion of the right graph for small values of $\log_2 p$.

For all schemes we used a Hamming weight of $h = 64$ to generate the secret key data, we used a security level of $k = 80$ bits of security, a standard deviation of $\sigma = 3.2$ for the rounded Gaussians, a tolerance factor of $\zeta = 8$ and a ring constant of $c_m = 1.3$. These are all consistent with the prior estimates for parameters given in [9]. The use of a small ring constant can be justified by either selecting $\phi(m)$ to be a power of two, or selecting m to be prime, as explained in [4]. As a general conclusion we find that for FV and YASHE the use of modulus switching to lower levels results in slightly bigger parameters to start for large values of L; approximately a factor of two for $L = 20$ or 30. But as a homomorphic calculation progresses this benefit will drop away, leaving, for most calculations, the variant in which modulus switching is applied the most efficient. Thus in what follows we assume that modulus switching is applied in all schemes.

Firstly examine the graphs in Figures 4 and 5. We see that for a fixed number of levels and very small plaintext moduli the most efficient scheme seems to be YASHE. However, quite rapidly, as the plaintext modulus increases the BGV scheme quickly outperforms all other schemes. In particular for the important case of the SPDZ MPC system [4] which requires an SHE scheme supporting circuits of multiplicative depth one, i.e. $L = 2$, for a large plaintext modulus p, the BGV scheme is seen to be the most efficient.

Examining Fig. 6 we see that if we fix the prime and just increase the number of levels then the choice of which is the better scheme is be very consistent. Thus one is led to conclude that the main choice of which scheme to adopt depends on the plaintext modulus, where one selects YASHE for very small plaintext moduli and BGV for larger plaintext moduli.

Acknowledgements

This work has been supported in part by an ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO and by the European Union's H2020 Programme under grant agree-

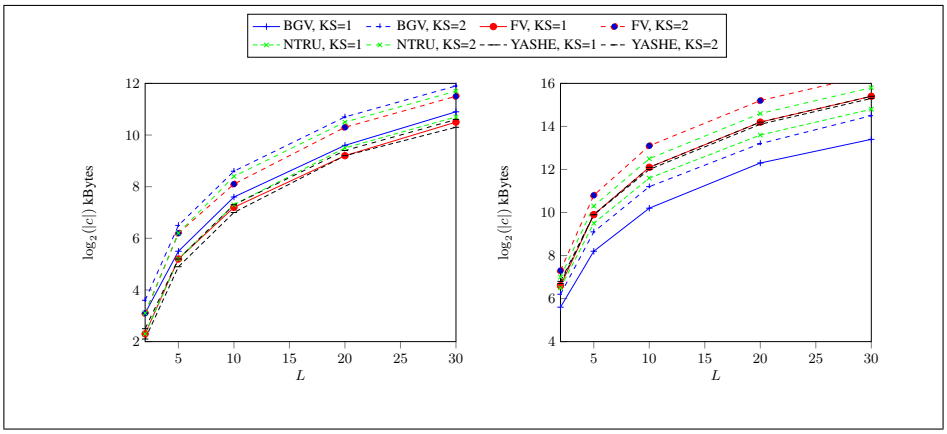

Fig. 6: Size of required ciphertext for various values of L when $p = 2$ and $p \approx 2^{32}$.

ment number ICT-644209. The authors would like to thank Steven Galbraith for comments on an earlier version of this manuscript.

References

1. J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In M. Stam, editor, *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2013.
2. Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Safavi-Naini and Canetti [16], pages 868–886.
3. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science (ITCS'12)*, 2012. Available at `http://eprint.iacr.org/2011/277`.
4. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini and Canetti [16], pages 643–662.
5. Y. Doröz, Y. Hu, and B. Sunar. Homomorphic AES evaluation using the modified LTV scheme. *Des. Codes and Cryptography*, XXXX:XXXX–XXXX, 2015.
6. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
7. C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.
8. C. Gentry, S. Halevi, and N. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. Full version at `http://eprint.iacr.org/2011/566`.
9. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In Safavi-Naini and Canetti [16], pages 850–867.
10. P. Kirchner and P. Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 43–62. Springer, 2015.

11. K. Lauter, M. Naehrig, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, pages 113–124. ACM, 2011.
12. T. Lepoint and M. Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In D. Pointcheval and D. Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, volume 8469 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2014.
13. R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
14. A. Lòpez-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*. ACM, 2012.
15. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, 2010.
16. R. Safavi-Naini and R. Canetti, editors. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.
17. N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.