Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers

Jean-Sébastien Coron[1], David Naccache[2], and Mehdi Tibouchi[3]

[1] Université du Luxembourg
jean-sebastien.coron@uni.lu
[2] École normale supérieure
david.naccache@ens.fr
[3] NTT Information Sharing Platform Laboratories
tibouchi.mehdi@lab.ntt.co.jp

Abstract. We describe a compression technique that reduces the public key size of van Dijk, Gentry, Halevi and Vaikuntanathan's (DGHV) fully homomorphic scheme over the integers from $\tilde{\mathcal{O}}(\lambda^7)$ to $\tilde{\mathcal{O}}(\lambda^5)$. Our variant remains semantically secure, but in the random oracle model. We obtain an implementation of the full scheme with a 10.1 MB public key instead of 802 MB using similar parameters as in [8]. Additionally we show how to extend the quadratic encryption technique of [8] to higher degrees, to obtain a shorter public-key for the basic scheme.
This paper also describes a new modulus switching technique for the DGHV scheme that enables to use the new FHE framework without bootstrapping from Brakerski, Gentry and Vaikuntanathan with the DGHV scheme. Finally we describe an improved attack against the Approximate GCD Problem on which the DGHV scheme is based, with complexity $\tilde{\mathcal{O}}(2^\rho)$ instead of $\tilde{\mathcal{O}}(2^{3\rho/2})$.

1 Introduction

Fully Homomorphic Encryption. An encryption scheme is said to be fully homomorphic when it is possible to perform implicit plaintext additions and multiplications while manipulating only ciphertexts.

The first construction of a fully homomorphic scheme was described by Gentry in [10]. Gentry first obtained a "somewhat homomorphic" scheme, supporting only a limited number of ciphertext multiplications due to the fact that ciphertext contain a certain amount of "noise" which increases with every multiplication, and that decryption fails when noise size passes a certain bound. As a result, in the somewhat homomorphic scheme, the functions that can be homomorphically evaluated on ciphertexts are polynomials of small, bounded degree. The second step in Gentry's framework consists in "squashing" the decryption procedure so that it can be expressed as a low degree polynomial in the bits of the ciphertext and the secret key. Then, Gentry's key idea, called "bootstrapping", is to evaluate this decryption polynomial not on the ciphertext bits and the secret-key bits (which would yield the plaintext), but homomorphically on the encryption of those bits, which gives another ciphertext of the same plaintext. If the degree of the decryption polynomial is small enough, the noise in the new ciphertext can become smaller than it was the original ciphertext, so that this new ciphertext can be used again in a subsequent homomorphic operation (either addition or multiplication). Using this "ciphertext refresh" procedure the number of permissible homomorphic operations becomes unlimited and one obtains a fully homomorphic encryption scheme. To date, three different fully homomorphic schemes are known:

1. Gentry's original scheme [10], based on ideal lattices. Gentry and Halevi described in [11] the first implementation of Gentry's scheme, using many clever optimizations, including some suggested in

a previous work by Smart and Vercauteren [15]. For their most secure setting (claiming 72 bit security) the authors report a public key size of 2.3 GB and a ciphertext refresh procedure taking 30 minutes on a high-end workstation.

2. van Dijk, Gentry, Halevi and Vaikuntanathan's (DGHV) scheme over the integers [9]. This scheme is conceptually simpler than Gentry's scheme, because it operates on integers instead of ideal lattices. Recently it was shown [8] how to reduce the public key size by storing only a small subset of the original public key and generating the full public key on the fly by combining the elements in the small subset multiplicatively. Using some of the optimizations from [11], the authors of [8] report similar performances: a 802 MB public key and a ciphertext refresh in 14 minutes.
3. Brakerski and Vaikuntanathan's scheme based on the Learning with Errors (LWE) and Ring Learning with Errors (RLWE) problems [3, 4]. The authors introduce a new dimension reduction technique and a new modulus switching technique to shorten the ciphertext and reduce the decryption complexity. A partial implementation is described in [12], without the fully homomorphic capability.

Recently Brakerski, Gentry and Vaikuntanathan introduced a remarkable new FHE framework, in which the noise ceiling increases only linearly with the multiplicative level instead of exponentially [5]; this implies that bootstrapping is no longer necessary to achieve fully homomorphic encryption. This new framework has the potential to significantly improve the practical FHE performance. The new framework is based on Brakerski and Vaikuntanathan's scheme [3, 4], and more specifically on their new modulus switching technique, which efficiently transforms a ciphertext encrypted under a certain modulus p into a ciphertext under a different modulus p' but with reduced noise.

Public Key Compression. The first of our contributions is a technique to reduce the public key size of DGHV-like schemes [9] by several orders of magnitude. In the DGHV scheme the public key is a set of integers of the form:

$$x_i = q_i \cdot p + r_i$$

where p is the secret-key of η bits, q_i is a large random integer of $\gamma - \eta$ bits, and r_i is a small random integer of ρ bits. The scheme's semantic security is based on the Approximate GCD Problem: given a polynomial number of x_i's, recover the secret p. To avoid lattice attacks, the bit-size γ of the x_i's must be very large: [8] takes $\gamma \simeq 2 \cdot 10^7$ for $\eta = 2652$ and $\rho = 39$, and the full public key claims a 802 MB storage.

Our technique proceeds as follows. First generate the secret-key p. Then, use a pseudo-random number generator f with public random seed se to generate a set of γ-bit integers χ_i (*i.e.* the χ_i's are of the same bit-size as the x_i's). Finally, compute small corrections δ_i to the χ_i's such that $x_i = \chi_i - \delta_i$ is small modulo p, and store only the small corrections δ_i in the public key, instead of the full x_i's. Knowing the PRNG seed se and the δ_i's is sufficient to recover the x_i's.

Therefore instead of storing a set of large γ-bit integers we only have to store a set of much smaller η-bit integers, where η is the bit size of p. The new technique is fully compatible with the DGHV variant described in [8]; with the previous set of parameters from [8] one obtains a public key size of 4.6 MB for the full implementation, instead of the 802 MB required in [8]! The technique can be seen as generating the $\gamma - \eta$ most significant bits of the x_i's with a pseudo-random number generator, and then using the secret key p to fix the η remaining bits so that $x_i \bmod p$ is small. While different, this is somewhat reminiscent of Lenstra's technique [13] for generating an RSA modulus with a predetermined portion.

As an aside, we briefly explain how a similar technique can also be applied to Brakerski and Vaikuntanathan's scheme. However, our technique does not seem to adapt readily to Gentry's scheme.

Under our variant, the encryption scheme can still be proved semantically secure under the Approximate GCD assumption, albeit in the random oracle model. This holds for both the original DGHV scheme form [9] and the variant described in [8] in which the public key elements are first combined multiplicatively to generate the full public key. Unlike [8, 9], we need the random oracle model in order to apply the leftover hash lemma in our variant, because the seed of the PRNG is known to the attacker (as part of the public key).

We report the result of an implementation of the new variant with the fully homomorphic capability. As in [8] we use the variant with noise-free $x_0 = q_0 \cdot p$. We also update the parameters from [8] to take into the account the improved attack from Chen and Nguyen against the Approximate GCD problem [7]. We obtain a level of efficiency very similar to [8] but with a 10.1 MB public key instead of a 802 MB one. The source code of this implementation is publicly available [18].

Extension to Higher Degrees. Various techniques have been proposed in [8] to reduce the public key size and increase the efficiency of the DGHV scheme, the most important of which is to use a *quadratic form* instead of a linear form for masking the message when computing a ciphertext. More precisely, ciphertexts are computed as:

$$c^* = m + 2r + 2 \sum_{1 \leq i,j \leq \beta} b_{ij} \cdot x_{i,0} \cdot x_{j,1} \mod x_0$$

which is quadratic in the public key elements $x_{i,b}$ instead of linear as in the original DGHV scheme. The authors show that the scheme remains semantically secure; the key ingredient is to prove that a certain family of quadratic hash functions is close enough to being pairwise independent, so that the leftover hash lemma can still be applied. The main benefit is a significant reduction in public key size, from $\tau = \tilde{\mathcal{O}}(\lambda^3)$ elements x_i down to $2\beta = \tilde{\mathcal{O}}(\lambda^{1.5})$ elements $x_{i,b}$. In this paper we prove that the natural extension of this quadratic encryption technique to to cubic forms, and more generally forms of arbitrary fixed degree d, remains secure, making it possible to further reduce the public key size.

Modulus Switching and Leveled DGHV Scheme. As a third contribution, we show how to adapt Brakerski, Gentry and Vaikuntanathan's (BGV) new FHE framework [5] to the DGHV scheme over the integers. Under the BGV framework the noise ceiling increases only linearly with multiplicative depth, instead of exponentially. This enables to get a FHE scheme without the costly bootstrapping procedure.

More precisely the new BGV framework is described in [5] with Brakerski and Vaikuntanathan's scheme [3], and the key technical tool is the modulus-switching technique of [3] that transforms a ciphertext $\boldsymbol{c}$ modulo p into a ciphertext $\boldsymbol{c'}$ modulo p' simply by scaling by p'/p and rounding appropriately. This allows to reduce the ciphertext noise by a factor close to p'/p without knowing the secret-key and without bootstrapping. However the modulus switching technique cannot directly apply to DGHV since in DGHV the moduli p and p' are secret. In this paper we explain how this modulus-switching technique can be adapted to DGHV, so as to apply the new BGV framework. We show that the resulting FHE scheme remains semantically secure, albeit under a stronger assumption. We also describe an implementation, showing that the new BGV framework can be applied in practice.

Improved Attack against the Approximate-GCD problem. Finally we consider the security of the Approximate GCD Problem *without* noise-free $x_0 = q_0 \cdot p$. In our leveled DGHV variant under the BGV framework the size of the secret p can become much smaller than in the original Gentry framework ($\eta \simeq 180$ bits for the lowest p in the ladder, instead of $\eta = 2652$ bits in [8]). This implies that the noise-free variant $x_0 = q_0 \cdot p$ cannot be used, since otherwise the prime factor p could easily be extracted using the Elliptic Curve Method for integer factorization [14]. Therefore one must consider the security of the Approximate GCD Problem *without* noise-free x_0. The recent attack by Chen and Nguyen [7] against the Approximate GCD Problem *with* noise-free x_0 has complexity $\tilde{\mathcal{O}}(2^{\rho/2})$, instead of the $\tilde{\mathcal{O}}(2^{\rho})$ naive attack; as noted by the authors, this immediately yields an $\tilde{\mathcal{O}}(2^{3\rho/2})$ attack against the Approximate GCD Problem without noise-free x_0, instead of $\tilde{\mathcal{O}}(2^{2\rho})$ for the naive attack. In this paper we exhibit an improved attack with complexity $\tilde{\mathcal{O}}(2^{\rho})$. We also describe an implementation showing that this new attack is indeed an improvement in practice.

2 The DGHV Scheme over the Integers.

We first recall the somewhat homomorphic encryption scheme described by van Dijk, Gentry, Halevi and Vaikuntanathan (DGHV) in [9]. For a real number x, we denote by $\lceil x \rceil$, $\lfloor x \rfloor$ and $\lceil x \rfloor$ the rounding of x up, down, or to the nearest integer. For integers z, p we denote the reduction of z modulo p by $[z]_p$ with $-p/2 < [z]_p \leq p/2$, and by $\langle z \rangle_p$ with $0 \leq \langle z \rangle_p < p$. Given the security parameter λ, the following parameters are used:

- γ is the bit-length of the x_i's,
- η is the bit-length of the secret key p,
- ρ is the bit-length of the noise r_i,
- τ is the number of x_i's in the public key,
- ρ' is a secondary noise parameter used for encryption.

For a specific η-bit odd integer p, we use the following distribution over γ-bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) = \left\{ \text{ Choose } q \leftarrow \mathbb{Z} \cap [0, 2^{\gamma}/p),\ r \leftarrow \mathbb{Z} \cap (-2^{\rho}, 2^{\rho}) :\ \text{Output } x = q \cdot p + r \right\}$$

$\mathsf{DGHV.KeyGen}(1^{\lambda})$. Generate a random prime integer p of size η bits. For $0 \leq i \leq \tau$ sample $x_i \leftarrow \mathcal{D}_{\gamma,\rho}(p)$. Relabel the x_i's so that x_0 is the largest. Restart unless x_0 is odd and $[x_0]_p$ is even. Let $pk = (x_0, x_1, \ldots x_{\tau})$ and $sk = p$.

$\mathsf{DGHV.Encrypt}(pk, m \in \{0, 1\})$. Choose a random subset $S \subseteq \{1, 2, \ldots, \tau\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output the ciphertext:

$$c = \left[m + 2r + 2 \sum_{i \in S} x_i \right]_{x_0} \tag{1}$$

$\mathsf{DGHV.Evaluate}(pk, C, c_1, \ldots, c_t)$: given the circuit C with t input bits, and t ciphertexts c_i, apply the addition and multiplication gates of C to the ciphertexts, performing all the additions and multiplications over the integers, and return the resulting integer.

$\mathsf{DGHV.Decrypt}(sk, c)$. Output $m \leftarrow [c]_p \bmod 2$.

This completes the description of the scheme. As shown in [9] this scheme is somewhat homomorphic, *i.e.* a limited number of homomorphic operations can be performed on ciphertexts. More precisely given two ciphertexts $c = q \cdot p + 2r + m$ and $c' = q' \cdot p + 2r' + m'$ where r and r' are ρ'-bit integers, the ciphertext $c + c'$ is an encryption of $m + m' \bmod 2$ with $(\rho' + 1)$-bit noise and the ciphertext $c \cdot c'$ is an encryption of $m \cdot m'$ with noise $\simeq 2\rho'$. Since the ciphertext noise must remain smaller than p for correct decryption, the scheme allows roughly η/ρ' multiplications on ciphertexts. As shown in [9] the scheme is semantically secure under the Approximate GCD assumption.

Definition 1 (Approximate GCD). *The (ρ, η, γ)-Approximate GCD Problem is: For a random η-bit odd integer p, given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$, output p.*

3 The New DGHV Public Key Compression Technique

We describe our technique using the variant with noise free $x_0 = q_0 \cdot p$, as suggested in [9] and implemented in [8]. We only describe the basic scheme; we refer to Appendix A for a complete description of the fully homomorphic scheme.

3.1 Description

$\mathsf{KeyGen}(1^\lambda)$. Generate a random prime integer p of size η bits. Pick a random odd integer $q_0 \in [0, 2^\gamma/p)$ and let $x_0 = q_0 \cdot p$. Initialize a pseudo-random number generator f with a random seed se. Use $f(\mathsf{se})$ to generate a set of integers $\chi_i \in [0, 2^\gamma)$ for $1 \le i \le \tau$. For all $1 \le i \le \tau$ compute:

$$\delta_i = \langle \chi_i \rangle_p + \xi_i \cdot p - r_i$$

where $r_i \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ and $\xi_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda+\eta}/p)$. For all $1 \le i \le \tau$ compute:

$$x_i = \chi_i - \delta_i \tag{2}$$

Let $pk = (\mathsf{se}, x_0, \delta_1, \ldots, \delta_\tau)$ and $sk = p$.

$\mathsf{Encrypt}(pk, m \in \{0, 1\})$: use $f(\mathsf{se})$ to recover the integers χ_i and let $x_i = \chi_i - \delta_i$ for all $1 \le i \le \tau$. Choose a random integer vector $\boldsymbol{b} = (b_i)_{1 \le i \le \tau} \in [0, 2^\alpha)^\tau$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$. Output the ciphertext:

$$c = m + 2r + 2\sum_{i=1}^{\tau} b_i \cdot x_i \mod x_0$$

$\mathsf{Evaluate}(pk, C, c_1, \ldots, c_t)$ and $\mathsf{Decrypt}(sk, c)$: same as in the original DGHV scheme, except that ciphertexts are reduced modulo x_0.

This completes the description of our variant. We have the following constraints on the scheme parameters:

- $\rho = \omega(\log \lambda)$ to avoid brute force attack on the noise,
- $\eta \ge \rho \cdot \Theta(\lambda \log^2 \lambda)$ in order to support homomorphic operations for evaluating the "squashed decryption circuit" (see [9]),
- $\gamma = \omega(\eta^2 \cdot \log \lambda)$ in order to thwart lattice-based attacks against the Approximate GCD problem (see [8, 9]),

- $\alpha \cdot \tau \geq \gamma + \omega(\log \lambda)$ in order to apply the left-over hash lemma (see [8, 9]).
- $\eta \geq \rho + \alpha + 2 + \log_2 \tau$ for correct decryption of a ciphertext,
- $\rho' = \alpha + \rho + \omega(\log \lambda)$ for the secondary noise parameter.

To satisfy the above constraints one can take $\rho = \lambda$, $\eta = \tilde{\mathcal{O}}(\lambda^2)$, $\gamma = \tilde{\mathcal{O}}(\lambda^5)$, $\alpha = \tilde{\mathcal{O}}(\lambda^2)$, $\tau = \tilde{\mathcal{O}}(\lambda^3)$ and $\rho' = \tilde{\mathcal{O}}(\lambda^2)$. The main difference with the original DGHV scheme is that instead of storing the large x_i's in the public key we only store the much smaller δ_i's. The new public key for the somewhat homomorphic scheme has size $\gamma + \tau \cdot (\eta + \lambda) = \tilde{\mathcal{O}}(\lambda^5)$ instead of $(\tau + 1) \cdot \gamma = \tilde{\mathcal{O}}(\lambda^8)$.

Remark 1. We can also compress x_0 by letting $x_0 = \chi_0 - \delta_0$ and storing only $\delta_0 = \langle \chi_0 \rangle_p + \xi_0 \cdot p$ in the public-key.

Remark 2. In the description above we add a random multiple of p to $\langle \chi_i \rangle_p$ in the δ_i's. This is done to obtain a proof of semantic security in the random oracle model (see below). However the scheme seems heuristically secure without adding the random multiple.

Remark 3. For encryption the integers x_i need not be stored in memory as they can be generated on the fly when computing the subset sum.

3.2 Semantic Security

Theorem 1. *The previous encryption scheme is semantically secure under the Approximate GCD assumption with noise-free $x_0 = q_0 \cdot p$, in the random oracle model.*

Proof. The proof is almost the same as in [9]. For simplicity we consider the variant without noise-free $x_0 = q_0 \cdot p$, as in [9]; the extension to the noise-free variant is straightforward. More precisely, we assume that x_0 is first generated as the other x_i's; then as described in Section 2 one relabels the x_i's so that x_0 is the largest, and restarts unless x_0 is odd and $[x_0]_p$ is even. Given a random oracle $H : \{0,1\}^* \to \mathbb{Z} \cap [0, 2^\gamma)$, we assume that the pseudo-random number generation of the χ_i's is defined as follows:

$$\chi_i = H(\mathsf{se} \,\|i) \quad \text{for all } 0 \leq i \leq \tau$$

and we show that the integers x_i's generated in (2) have a distribution statistically close to their distribution in the original DGHV scheme.

As in [9] given an attacker $\mathcal{A}$ that breaks the semantic security of the scheme, one constructs a solver $\mathcal{B}$ for the Approximate GCD Problem. Our proof differs from [9] only in the first step: the creation of the public key.

Step 1: Creating the public key. The solver $\mathcal{B}$ begins by constructing a public key for the scheme. $\mathcal{B}$ obtains $\tau + 1$ samples $x_0, \ldots, x_\tau \leftarrow \mathcal{D}_{\gamma,\rho}(p)$. $\mathcal{B}$ generates a random seed se and programs the random oracle in the following way for all $0 \leq i \leq \tau$:

$$H(\mathsf{se} \,\|i) = x_i + \delta_i$$

where $\delta_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda+\eta})$. For other inputs the random oracle is simulated in the usual way, *i.e.* by generating a random output in $\mathbb{Z} \cap [0, 2^\gamma)$ for every fresh input. Finally $\mathcal{B}$ relabels the x_i's so that x_0 is the largest. $\mathcal{B}$ restarts unless x_0 is odd. $\mathcal{B}$ then outputs a public key $pk = (\mathsf{se}, \delta_0, \ldots, \delta_\tau)$.

The following Lemma shows that if $[x_0]_p$ happens to be even then the distribution of the public key is statistically close to that of the scheme.

Lemma 1. *The following two distributions have statistical distance $\mathcal{O}(2^{-\lambda})$:*

$$\begin{aligned}\mathcal{D} = \big\{(\chi,\delta,x);\ & \chi \leftarrow \mathbb{Z}\cap[0,2^{\gamma}),\ \delta = \langle\chi\rangle_p + \xi\cdot p - r,\ x = \chi - \delta,\\ & \xi \leftarrow \mathbb{Z}\cap[0,2^{\lambda+\eta}/p),\ r \leftarrow \mathbb{Z}\cap(-2^{\rho},2^{\rho})\big\}\\ \mathcal{D}' = \big\{(\chi,\delta,x);\ & x = q\cdot p + r,\ \delta \leftarrow \mathbb{Z}\cap[0,2^{\lambda+\eta}),\ \chi = x+\delta,\\ & q \leftarrow \mathbb{Z}\cap[0,2^{\gamma}/p),\ r \leftarrow \mathbb{Z}\cap(-2^{\rho},2^{\rho})\big\}\end{aligned}$$

The distribution of the x_i's after Step 1 is the same as the one in the proof of semantic security from [9], and therefore the rest of the proof is the same as in [9]. □

3.3 Proof of Lemma 1

Given X and Y two random variables over a finite set S, we denote by

$$d(X,Y) := \frac{1}{2}\sum_{s\in S}\big|\Pr[X=s]-\Pr[Y=s]\big|$$

the statistical distance between X and Y. We say that X and Y are statistically indistinguishable if $d(X,Y)$ is a negligible function of the security parameter.

We denote by U_S the uniform distribution over a finite set S. Given a finite set $\mathit{\Delta}$ such that $\mathit{\Delta}\cap S = \varnothing$, we have:

$$d(U_S,U_{S\cup\mathit{\Delta}}) = \frac{1}{2}\sum_{s\in S}\left|\frac{1}{|S|}-\frac{1}{|S|+|\mathit{\Delta}|}\right| + \frac{1}{2}\sum_{s\in\mathit{\Delta}}\left|0-\frac{1}{|S|+|\mathit{\Delta}|}\right| = \frac{|\mathit{\Delta}|}{|S|+|\mathit{\Delta}|}$$

We let $q_0 = \lfloor 2^{\gamma}/p\rfloor$. We modify $\mathcal{D}$ by generating $\chi \leftarrow \mathbb{Z}\cap[0,p\cdot q_0)$ instead of $\chi\leftarrow\mathbb{Z}\cap[0,2^{\gamma})$ and denote by $\mathcal{D}_1$ the corresponding distribution. The statistical distance between $\mathcal{D}$ and $\mathcal{D}_1$ is at most $p/2^{\gamma}\le 2^{\eta-\gamma}$. Equivalently we let $\chi = q'\cdot p+\alpha$ where $q'\leftarrow\mathbb{Z}\cap[0,q_0)$ and $\alpha\leftarrow\mathbb{Z}\cap[0,p)$. This gives:

$$\begin{aligned}\mathcal{D}_1 = \big\{(\chi,\delta,x);\ & \chi = q'\cdot p+\alpha,\ \delta = \alpha+\xi\cdot p - r,\ x = \chi-\delta,\\ & q'\leftarrow\mathbb{Z}\cap[0,q_0),\ \alpha\leftarrow\mathbb{Z}\cap[0,p),\ \xi\leftarrow\mathbb{Z}\cap[0,2^{\lambda+\eta}/p),\ r\leftarrow\mathbb{Z}\cap(-2^{\rho},2^{\rho})\big\}\\ = \{(\chi,\delta,x);\ & x = (q'-\xi)\cdot p + r,\ \delta = \alpha+\xi\cdot p - r,\ \chi = x+\delta,\\ & q'\leftarrow\mathbb{Z}\cap[0,q_0),\ \alpha\leftarrow\mathbb{Z}\cap[0,p),\ \xi\leftarrow\mathbb{Z}\cap[0,2^{\lambda+\eta}/p),\ r\leftarrow\mathbb{Z}\cap(-2^{\rho},2^{\rho})\}\end{aligned}$$

We first consider the distribution:

$$\mathcal{D}_q = \Big\{q'-\xi;\ q'\leftarrow\mathbb{Z}\cap[0,q_0),\ \xi\leftarrow\mathbb{Z}\cap[0,2^{\lambda+\eta}/p)\Big\}$$

We have:

$$\begin{aligned}d(\mathcal{D}_q,U_{\mathbb{Z}\cap[0,2^{\gamma}/p)}) &\le d(\mathcal{D}_q,U_{\mathbb{Z}\cap[0,q_0)}) + d(U_{\mathbb{Z}\cap[0,q_0)},U_{\mathbb{Z}\cap[0,2^{\gamma}/p)})\\ &\le \frac{2^{\lambda+\eta+1}/p}{q_0}+\frac{1}{q_0}\le 2^{\lambda+\eta-\gamma+2}\end{aligned}$$

Similarly we consider the distribution:

$$\begin{aligned}\mathcal{D}_\delta &= \Big\{\alpha+\xi\cdot p-r;\ \alpha\leftarrow\mathbb{Z}\cap[0,p),\ \xi\leftarrow\mathbb{Z}\cap[0,2^{\lambda+\eta}/p),\ r\leftarrow\mathbb{Z}\cap(-2^{\rho},2^{\rho})\Big\}\\ &= \big\{\delta'-r;\ \delta'\leftarrow\mathbb{Z}\cap[0,p\cdot\xi_0),\ r\leftarrow\mathbb{Z}\cap(-2^{\rho},2^{\rho})\big\}\end{aligned}$$

where $\xi_0 = \lceil 2^{\lambda+\eta}/p \rceil$, and we have:

$$\begin{aligned} d(\mathcal{D}_\delta, U_{\mathbb{Z}\cap[0,2^{\lambda+\eta})}) &\leq d(\mathcal{D}_\delta, U_{\mathbb{Z}\cap[0,p\cdot\xi_0)}) + d(U_{\mathbb{Z}\cap[0,p\cdot\xi_0)}, U_{\mathbb{Z}\cap[0,2^{\lambda+\eta})}) \\ &\leq \frac{2^{\rho+2}}{p\cdot\xi_0} + \frac{p}{2^{\lambda+\eta}} \leq 2^{\rho-\eta-\lambda+2} + 2^{-\lambda} \end{aligned}$$

Finally, we have:

$$d(\mathcal{D}_1, \mathcal{D}') \leq d(\mathcal{D}_q, U_{\mathbb{Z}\cap[0,2^\gamma/p)}) + d(\mathcal{D}_\delta, U_{\mathbb{Z}\cap[0,2^{\lambda+\eta})}) \leq 2^{\lambda+\eta-\gamma+2} + 2^{\rho-\eta-\lambda+2} + 2^{-\lambda}$$

and:

$$d(\mathcal{D}, \mathcal{D}') \leq d(\mathcal{D}, \mathcal{D}_1) + d(\mathcal{D}_1, \mathcal{D}') \leq 2^{\eta-\gamma} + 2^{\lambda+\eta-\gamma+2} + 2^{\rho-\eta-\lambda+2} + 2^{-\lambda}$$

Given the constraints on the scheme's parameters, this statistical distance is $\mathcal{O}(2^{-\lambda})$.

3.4 Extension to Other Fully Homomorphic Schemes

We briefly explain how the new compression technique can be extended to Brakerski and Vaikuntanathan's scheme based on the Learning with Errors (LWE) problem [3, 4]. We also explain why our technique is not readily applicable to Gentry's scheme.

Brakerski and Vaikuntanathan's Scheme. The public key is a set of ciphertexts of the form:

$$x_i = (\boldsymbol{a_i}, \langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + r_i \bmod q)$$

where $\boldsymbol{a_i}$ is a random element in $\mathbb{Z}_q^n$, $\boldsymbol{s} \in \mathbb{Z}_q^n$ is the secret key, and r_i is a small random noise in $\mathbb{Z}_q$. Therefore to compress the public key it suffices to generate the $\boldsymbol{a_i}$'s pseudo-randomly and store only the one-dimensional terms $\langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + r_i \bmod q$.

However the technique does not seem to extend to the Ring Learning with Errors (RLWE) scheme [4]. A ciphertext is a pair of ring elements $(a, a \cdot s + r)$, where a is a random element in the ring R, $s \in R$ is the secret key, and r is a small random noise in R. One can generate the first element a pseudo-randomly but this only halves the ciphertext size.

Gentry's Scheme. The technique doesn't seem to adapt readily to Gentry's scheme. If we consider for example the variant by Gentry and Halevi [11], the public key of the underlying somewhat homomorphic scheme is relatively short: it is a pair (d, r) of $(n \cdot t)$-bit integers (with $t = 380$ and $n = 32768$ at the "large" security level). What makes the public key of the fully homomorphic scheme rather large is that it contains encryptions of secret bits s_i used in the squashed decryption circuit.

An encryption of a bit m in the Gentry-Halevi scheme is computed as:

$$c = m + 2u(r) \bmod d$$

where (d, r) is the public key of the underlying scheme, and u is a random polynomial of degree n with coefficients in $\{-1, 0, 1\}$ (with 0 being chosen with higher probability than ±1, but we will ignore this fact for a moment).

A natural generalization of our technique would be to compute the encryptions of the secret bits s_i as follows: generate pseudo-random numbers $\chi_i \in \mathbb{Z}_d$, choose polynomials u_i as above such that the differences $\delta_i = [\chi_i - s_i - 2u_i(r)]_d$ are small, and publish the δ_i's in the public key. The problem is that the δ_i's cannot be too small if the polynomials u_i are to exist at all.

Indeed, we have $s_i + 2u_i(r) + \delta_i \equiv \chi_i \pmod{d}$, so we must pick δ_i large enough that all elements of $\mathbb{Z}_d$ can be represented in that form. If we bound δ_i by $|\delta_i| < D/2$, there are at most $3^n \cdot D$ elements of $\mathbb{Z}_d$ represented as the left-hand side: hence, we must have $D > d/3^n = (2^{380}/3)^n$. This implies that, at best, the technique can only reduce the bit length of the public key by a factor of $(\log 3)/(380 \log 2) \approx 0.4\%$ (and even less if we take into account the fact that u_i should be chosen relatively sparse with good probability).

4 Extension of DGHV Encryption to Higher Degrees

Various techniques have recently been proposed in [8] to reduce the public key size and increase the efficiency of the DGHV scheme, the most important of which is to use a *quadratic form* instead of a linear form for masking the message when computing a ciphertext. More precisely, one computes:

$$c^* = m + 2r + 2 \sum_{1 \le i,j \le \beta} b_{ij} \cdot x_{i,0} \cdot x_{j,1} \mod x_0$$

which is quadratic in the public key elements $x_{i,b}$ instead of linear as in equation (1); here the variant with noise-free $x_0 = q_0 \cdot p$ is used. The main benefit is a significant decrease in the public key size, from $\tau = \tilde{\mathcal{O}}(\lambda^3)$ elements x_i down to $2\beta = \tilde{\mathcal{O}}(\lambda^{1.5})$ elements $x_{i,b}$. Namely the constraint to apply the left-over hash lemma becomes $\alpha \cdot \beta^2 \ge \gamma + \omega(\log \lambda)$, so by taking $\alpha = \tilde{\mathcal{O}}(\lambda^2)$ one can take $\beta = \tilde{\mathcal{O}}(\lambda^{1.5})$. Combined with our compression technique the public-key size of the somewhat homomorphic scheme becomes $(2\beta + 1) \cdot (\eta + \lambda) = \tilde{\mathcal{O}}(\lambda^{3.5})$.

To prove that the scheme remains secure under this modified encryption procedure, the key point in [8] was to prove that the following family of functions $h\colon \{0, \ldots, 2^{\alpha-1}\}^{\beta^2} \to \mathbb{Z}_{q_0}$:

$$h(\boldsymbol{b}) = \sum_{1 \le i_1, i_2 \le \beta} b_{i_1 i_2} q_{i_1}^{(1)} q_{i_2}^{(2)} \mod q_0 \qquad \left(q_i^{(j)} \in \mathbb{Z}_{q_0}\right)$$

is close enough to being a pairwise independent (*i.e.* universal) hash function family (under suitable parameter choices), which in turn makes it possible to apply a variant of the leftover hash lemma.

In this section we show that it is possible to obtain further efficiency gains by using *cubic forms* instead, or more generally forms of higher degree d, if we can prove an analogue of the previous result for the family $\mathcal{H}_d$ of hash functions $h : \{0, \ldots, 2^{\alpha-1}\}^{\beta^d} \to \mathbb{Z}_q$ of the form:

$$h(\boldsymbol{b}) = \sum_{1 \le i_1, \ldots, i_d \le \beta} b_{i_1, \ldots, i_d} q_{i_1}^{(1)} \cdots q_{i_d}^{(d)} \mod q \qquad \left(q_i^{(j)} \in \mathbb{Z}_q\right)$$

Such a result also leads to the construction of extractors with relatively short seeds, which is an interesting fact in its own right.

We show that this hash function family is indeed close to being pairwise independent for suitable parameters. As in [8], we can prove this in the simpler case when $q = q_0$ is prime; the result then follows for all q_0 without small prime factors. The main result is as follows (we refer to [8] for the definition of ε-pairwise independence).

Theorem 2. *For an odd prime q, the hash function family $\mathcal{H}_d$ is ε-pairwise independent, with:*

$$\varepsilon = \frac{(d-1)(d-2)}{\sqrt{q}} + \frac{5d^{13/3}}{q} + \frac{(d-1)\cdot(2\beta)^d}{2^{\alpha\beta^{d-1}(\beta-2-2/\alpha))}}$$

Using the variant of the leftover hash lemma from [8], this proves the semantic security of the scheme for any encryption degree $d \geq 2$, with the condition $\alpha \cdot \beta^d \geq \gamma + \omega(\log \lambda)$. The constraint for correct decryption becomes $\eta \geq \rho \cdot d + \alpha + 2 + d \cdot \log_2 \beta$, and $\rho' = \rho \cdot d + \alpha + \omega(\log \lambda)$ for the secondary noise parameter. The public-key size for the somewhat homomorphic scheme becomes $(d \cdot \beta + 1) \cdot (\eta + \lambda)$. In particular by taking $\beta = 3$ and $d = \mathcal{O}(\log \lambda)$, we get a public-key size in $\tilde{\mathcal{O}}(\lambda^2)$ for the somewhat homomorphic scheme.

4.1 Proof of Theorem 2

We will use the following explicit version of the Lang-Weil bound for affine hypersurfaces.

Lemma 2 (Cafure & Matera [6]). *Let q be a prime and V be a hypersurface in $\mathbb{A}^N(\mathbb{F}_q)$ defined as the locus of zeros of a polynomial $P \in \mathbb{F}_q[x_1, \ldots, x_N]$ of total degree d. Then the number of $\mathbb{F}_q$-points of V is bounded as:*

$$\#V(\mathbb{F}_q) \leq dq^{N-1}$$

Moreover, if P is absolutely irreducible, then:

$$\#V(\mathbb{F}_q) \leq q^{N-1} + (d-1)(d-2)q^{N-3/2} + 5d^{13/3}q^{N-2}$$

Turning to the proof of Theorem 2, we observe that for each choice of $\boldsymbol{b} \neq \boldsymbol{b'}$, the probability $\Pr_{h \leftarrow \mathcal{H}_d}[h(\boldsymbol{b}) = h(\boldsymbol{b'})]$ can be expressed in terms of the number of $\mathbb{F}_q$-points of a hypersurface in $\mathbb{A}^{d\beta}\mathbb{F}_q$. More precisely, if we let $\boldsymbol{a} = \boldsymbol{b} - \boldsymbol{b'}$, we have:

$$\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] = \frac{1}{q^{d\beta}} \# \left\{ (u_i^{(j)}) \in \mathbb{F}_q^{d\times\beta} \ : \sum_{1 \leq i_1, \ldots, i_d \leq \beta} a_{i_1,\ldots,i_d} u_{i_1}^{(1)} \cdots u_{i_d}^{(d)} = 0 \right\} = \frac{1}{q^{d\beta}} \# V_{\boldsymbol{a}}(\mathbb{F}_q)$$

where $V_{\boldsymbol{a}}$ is the hypersurface of zeros of the polynomial:

$$P_{\boldsymbol{a}}(x_1^{(1)}, \ldots, x_\beta^{(1)}, \ldots, x_1^{(d)}, \ldots, x_\beta^{(d)}) = \sum_{1 \leq i_1, \ldots, i_d \leq \beta} a_{i_1,\ldots,i_d} x_{i_1}^{(1)} \cdots x_{i_d}^{(d)} \tag{3}$$

In particular, when $P_{\boldsymbol{a}}$ is absolutely irreducible, Lemma 2 gives:

$$\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] \leq \frac{1}{q} + \frac{(d-1)(d-2)}{q^{3/2}} + \frac{5d^{13/3}}{q^2}$$

Otherwise, we get:

$$\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] \leq \frac{d}{q}$$

Now, by definition, the hash function family $\mathcal{H}_d$ is ε'-pairwise independent for

$$\varepsilon' = \frac{\#\mathbb{F}_q}{\#X^2} \sum_{\boldsymbol{b} \neq \boldsymbol{b'}} \left(\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] - \frac{1}{\#\mathbb{F}_q} \right)$$

with $X = \{0, \dots, 2^{\alpha-1}\}^{\beta^d}$. By the above estimates, we get:

$$\begin{aligned}
\varepsilon' &\leq \frac{q}{\#X^2}\left(\sum_{\substack{\boldsymbol{b}\neq\boldsymbol{b'} \\ P_{\boldsymbol{a}} \text{ abs. irred.}}} \Big(\frac{(d-1)(d-2)}{q^{3/2}} + \frac{5d^{13/3}}{q^2}\Big) + \sum_{\substack{\boldsymbol{b}\neq\boldsymbol{b'} \\ P_{\boldsymbol{a}} \text{ not abs. irred.}}} \frac{d-1}{q} \right) \\
&\leq \frac{(d-1)(d-2)}{\sqrt{q}} + \frac{5d^{13/3}}{q} + (d-1)\cdot \frac{\#\{(\boldsymbol{b},\boldsymbol{b'}) \in X^2 \ : \ \boldsymbol{b} \neq \boldsymbol{b'} \text{ and } P_{\boldsymbol{a}} \text{ not abs. irred.}\}}{\#X^2}
\end{aligned}$$

Thus, if we denote by U_α the set of nonzero polynomials of the form (3) with coefficients in $\{-2^\alpha + 1, \dots, 2^\alpha - 1\}$ that are not absolutely irreducible, we get:

$$\varepsilon' \leq \frac{(d-1)(d-2)}{\sqrt{q}} + \frac{5d^{13/3}}{q} + (d-1)\cdot\frac{\#U_\alpha}{\#X} \tag{4}$$

Indeed, for any choice of $\boldsymbol{b'}$, there are at most $\#U_\alpha$ choices of $\boldsymbol{b}$ for which $P_{\boldsymbol{a}}$ is not absolutely irreducible.

So all we have to do is find a suitable upper bound for $\#U_\alpha$. Let:

$$P = \sum_{(i_1,\dots,i_d)} a_{i_1,\dots,i_d} x^{(1)}_{i_1} \cdots x^{(d)}_{i_d}$$

a polynomial in U_α. Since P is multilinear in $\underline{x}^{(1)}, \dots, \underline{x}^{(d)}$ and not absolutely irreducible, it factors over $\overline{\mathbb{F}}_q$ as a product of two polynomials Q, R which are multilinear in two complementary subsets of these variables. Assume for example, without loss of generality, that Q is multilinear in $\underline{x}^{(1)}, \dots, \underline{x}^{(\ell)}$ and R is multilinear in $\underline{x}^{(\ell+1)}, \dots, \underline{x}^{(d)}$, for some $1 \leq \ell \leq d-1$:

$$\begin{aligned}
Q &= \sum_{(i_1,\dots,i_\ell)} u_{i_1,\dots,i_\ell} x^{(1)}_{i_1} \cdots x^{(\ell)}_{i_\ell} \\
R &= \sum_{(i_{\ell+1},\dots,i_d)} v_{i_{\ell+1},\dots,i_d} x^{(\ell+1)}_{i_{\ell+1}} \cdots x^{(d)}_{i_d}
\end{aligned}$$

Clearly, for any multi-index $(i_1, \dots, i_d)$, we have $a_{i_1,\dots,i_d} = u_{i_1,\dots,i_\ell} \cdot v_{i_{\ell+1},\dots,i_d}$. Now assume further, still without loss of generality since $P \neq 0$, that $a_{1,\dots,1} \neq 0$. Then, all the coefficients of P are entirely determined by those of multi-index $(i_1, \dots, i_\ell, 1, \dots, 1)$ and those of multi-index $(1, \dots, 1, i_{\ell+1}, \dots, i_d)$. Indeed, for any multi-index $(i_1, \dots, i_d)$, we have:

$$a_{i_1,\dots,i_d} = u_{i_1,\dots,i_\ell} \cdot v_{i_{\ell+1},\dots,i_d} = \frac{(u_{i_1,\dots,i_\ell} v_{1,\dots,1}) \cdot (u_{1,\dots,1} v_{i_{\ell+1},\dots,i_d})}{u_{1,\dots,1} v_{1,\dots,1}} = \frac{a_{i_1,\dots,i_\ell,1,\dots,1} \cdot a_{1,\dots,1,i_{\ell+1},\dots,i_d}}{a_{1,\dots,1}}$$

This implies that there are at most $(2^{\alpha+1}-1)^{\beta^\ell} \cdot (2^{\alpha+1}-1)^{\beta^{d-\ell}} \leq 2^{2\beta^{d-1}(\alpha+1)}$ elements in U_α for this choice of two complementary subsets of variables and of a nonzero coefficient of P. Taking all possible choices into account, we get:

$$\#U_\alpha \leq \beta^d \cdot 2^d \cdot 2^{2\beta^{d-1}(\alpha+1)}$$

Plugging this bound into (4), we obtain the desired result.

5 Adaptation of the BGV Framework to the DGHV Scheme

5.1 The BGV Framework for Leveled FHE

In this section we first recall the new framework from Brakerski, Gentry and Vaikuntanathan (BGV) [5] for leveled fully homomorphic encryption. Under the BGV framework the noise ceiling increases only linearly with the multiplicative depth, instead of increasing exponentially. This implies that bootstrapping is no longer necessary to achieve fully homomorphic encryption. The new framework is based on the Brakerski and Vaikuntanathan RLWE scheme [3, 4]. The key technical tool is the modulus-switching technique from [3] that transforms a ciphertext $\boldsymbol{c}$ modulo p into a ciphertext $\boldsymbol{c'}$ modulo p' simply by scaling by p'/p and rounding appropriately; the noise is also reduced by a factor p'/p.

More precisely in [5] the decryption of a ciphertext vector $\boldsymbol{c}$ that encrypts m has the form $m = [\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_p \bmod 2$ where $\boldsymbol{s}$ is the secret-key. The term $[\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_p$ is called the "noise" associated to the ciphertext $\boldsymbol{c}$. The following lemma shows that from a ciphertext $\boldsymbol{c}$ encrypted under p one can efficiently obtain a new ciphertext $\boldsymbol{c'}$ under p', simply by multiplying every component by p'/p and rounding appropriately. The resulting noise is then essentially multiplied by the ratio p'/p. The modulus-switching technique is therefore a very lightweight procedure to reduce the ciphertext noise by a factor roughly p/p' without knowing the secret-key and without bootstrapping.

Lemma 3 (Modulus-Switching [3, 5]). *Let p and p' be two odd moduli. Let $\boldsymbol{c}$ by an integer vector and let $\boldsymbol{c'}$ be the integer vector closest to $(p'/p) \cdot \boldsymbol{c}$ such that $\boldsymbol{c'} = \boldsymbol{c} \pmod 2$. Then, for any $\boldsymbol{s}$ with $|[\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_p| < p/2 - (p/p') \cdot \ell_1(\boldsymbol{s})$, we have:*

$$[\langle \boldsymbol{c'}, \boldsymbol{s} \rangle]_{p'} = [\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_p \pmod 2 \quad \text{and} \quad |[\langle \boldsymbol{c'}, \boldsymbol{s} \rangle]_{p'}| < (p'/p) \cdot |[\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_p| + \ell_1(\boldsymbol{s})$$

where $\ell_1(\boldsymbol{s})$ is the ℓ_1-norm of $\boldsymbol{s}$.

In the original Gentry framework [10], the multiplication of two mod-p ciphertexts with noise size ρ gives a ciphertext with noise size $\simeq 2\rho$; after a second multiplication level the noise becomes $\simeq 4\rho$, then $\simeq 8\rho$ and so on; the noise size grows exponentially with the number of multiplication levels. The modulus p is a ceiling for correct decryption; therefore if the bit-size of p is $k \cdot \rho$, the noise ceiling is reached after only $\log_2 k$ levels of multiplication. Fully homomorphic encryption is achieved via bootstrapping, *i.e.* homomorphically evaluating the decryption polynomial to obtain a refreshed ciphertext.

The breakthrough idea in the BGV framework [5] is to apply the modulus-switching technique *after every multiplication level*, using a ladder of gradually decreasing moduli p_i. Start with two mod-p_1 ciphertexts with noise ρ; as previously after multiplication one gets a mod-p_1 ciphertext with noise 2ρ. Now switch to a new modulus p_2 such that $p_2/p_1 \simeq 2^{-\rho}$; after the switching one gets a mod-p_2 ciphertext with noise back to $2\rho - \rho = \rho$ again; one can continue by multiplying two mod-p_2 ciphertexts, obtain a 2ρ-noise mod-p_2 ciphertext and switch back to a ρ-noise mod-p_3 ciphertext, and so on. With a ladder of k moduli p_i of decreasing size $(k + 1) \cdot \rho, \ldots, 3\rho, 2\rho$ one can therefore perform k levels of multiplication instead of just $\log_2 k$. In other words the (largest) modulus size $(k + 1) \cdot \rho$ grows only *linearly* with the multiplicative depth; this is an exponential improvement.

As explained in [5], bootstrapping is no longer strictly necessary to achieve fully homomorphic encryption: namely one can always assume a polynomial upper-bound on the number L of multiplicative levels of the circuit to be evaluated homomorphically. However, bootstrapping is still an interesting operation as a bootstrapped scheme can perform homomorphic evaluations indefinitely without needing to specify at setup time a bound on the multiplicative depth. As shown in [5] bootstrapping becomes also more efficient asymptotically in the BGV framework.

5.2 Modulus-Switching for DGHV

The modulus-switching technique recalled in the previous section is a very lightweight procedure to reduce the ciphertext noise by a factor roughly p/p' without knowing the secret-key and without bootstrapping. However we cannot apply this technique directly to DGHV since in DGHV the moduli p and p' must remain secret.

We now describe a technique for switching moduli in DGHV. We proceed in two steps. Given as input a DGHV ciphertext $c = q \cdot p + r$, we first show in Lemma 4 how to obtain a "virtual" ciphertext of the form $c' = 2^k \cdot q' + r'$ with $[q'] = [q]_2$, given the bits s_i in the following subset-sum sharing of $2^k/p$:

$$\frac{2^k}{p} = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon \mod 2^{k+1}$$

where the y_i's have κ bits of precision after the binary point, with $|\varepsilon| \leq 2^{-\kappa}$. This is done by first "expanding" the initial ciphertext c using the y_i's, as in the "squashed decryption" procedure in [10], and then "collapsing" the expanded ciphertext into c', using the secret-key vector $\boldsymbol{s} = (s_i)$. However we cannot reveal $\boldsymbol{s}$ in clear, so instead we provide a DGHV encryption under p' of the secret-key bits s_i, as in the bootstrapped procedure. Then as showed in Lemma 5 the expanded ciphertext can be collapsed into a new ciphertext c'' under p' instead of p, for the same underlying plaintext; moreover as in the RLWE scheme the noise is reduced by a factor $\simeq p'/p$.

Lemma 4. *Let p be an odd integer. Let $c = q \cdot p + r$ be a ciphertext. Let k be an integer. Let $\kappa \in \mathbb{Z}$ be such that $|c| < 2^\kappa$. Let $\boldsymbol{y}$ be a vector of Θ numbers with κ bits of precision after the binary point, and let $\boldsymbol{s}$ be a vector of Θ bits such that $2^k/p = \langle \boldsymbol{s}, \boldsymbol{y} \rangle + \varepsilon \bmod 2^{k+1}$, where $|\varepsilon| \leq 2^{-\kappa}$. Let $\boldsymbol{c} = (\lfloor c \cdot y_i \rceil \bmod 2^{k+1})_{1 \leq i \leq \Theta}$. Let $c' = \langle \boldsymbol{s}, \boldsymbol{c} \rangle$. Then $c' = q' \cdot 2^k + r'$ with $[q']_2 = [q]_2$ and $r' = \lfloor r \cdot 2^k/p \rceil + \delta$ where $\delta \in \mathbb{Z}$ with $|\delta| \leq \Theta/2 + 2$.*

Proof. We have:

$$c' = \sum_{i=1}^{\Theta} s_i \lfloor c \cdot y_i \rceil + \Delta \cdot 2^{k+1} = \sum_{i=1}^{\Theta} s_i \cdot c \cdot y_i + \delta_1 + \Delta \cdot 2^{k+1}$$

for some $\Delta \in \mathbb{Z}$ and $|\delta_1| \leq \Theta/2$. Using $\langle \boldsymbol{s}, \boldsymbol{y} \rangle = 2^k/p - \varepsilon - \mu \cdot 2^{k+1}$ for some $\mu \in \mathbb{Z}$ this gives:

$$c' - \delta_1 - \Delta 2^{k+1} = c \cdot \left(\frac{2^k}{p} - \varepsilon - \mu \cdot 2^{k+1} \right) = q \cdot 2^k + r \cdot \frac{2^k}{p} - c \cdot \varepsilon - c \cdot \mu \cdot 2^{k+1}$$

Therefore we can write:

$$c' = q' \cdot 2^k + r'$$

where $[q']_2 = [q]_2$ and $r' = \lfloor r \cdot 2^k/p \rceil + \delta$ for some $\delta \in \mathbb{Z}$ with $|\delta| \leq \Theta/2 + 2$. □

As in [5], given a vector $\boldsymbol{x} \in [0, 2^{k+1}[^\Theta$ we write $\boldsymbol{x} = \sum_{i=0}^{k} 2^j \cdot \boldsymbol{u}_j$ where all the elements in vectors $\boldsymbol{u}_j$ are bits, and we define $\mathsf{BitDecomp}(\boldsymbol{x}, k) := (\boldsymbol{u}_0, \ldots, \boldsymbol{u}_k)$. Similarly given a vector $\boldsymbol{z} \in \mathbb{R}^\Theta$ we define $\mathsf{Powersof2}(\boldsymbol{z}, k) := (\boldsymbol{z}, 2 \cdot \boldsymbol{z}, \ldots, 2^k \cdot \boldsymbol{z})$. We have for any vectors $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\Big\langle \mathsf{BitDecomp}(\boldsymbol{x}, k), \mathsf{Powersof2}(\boldsymbol{z}, k) \Big\rangle = \langle \boldsymbol{x}, \boldsymbol{z} \rangle$$

The following lemma shows that given a ciphertext c encrypted under p and with noise r we can compute a new ciphertext c'' under p' with noise $r'' \simeq r \cdot p'/p$, by using an encryption $\boldsymbol{\sigma}$ under p' of the secret-key $\boldsymbol{s}$ corresponding to p.

Lemma 5. *Let p and p' be two odd integers. Let k be an integer such that $p' < 2^k$. Let $c = q \cdot p + r$ be a ciphertext. Let $\kappa \in \mathbb{Z}$ be such that $|c| < 2^\kappa$. Let $\boldsymbol{y}$ be a vector of Θ numbers with κ bits of precision after the binary point, and let $\boldsymbol{s}$ be a vector of Θ bits such that $2^k/p = \langle \boldsymbol{s}, \boldsymbol{y} \rangle + \varepsilon \bmod 2^{k+1}$, where $|\varepsilon| \le 2^{-\kappa}$. Let $\boldsymbol{\sigma} = p' \cdot \boldsymbol{q} + \boldsymbol{r} + \lfloor \boldsymbol{s'} \cdot p'/2^{k+1} \rceil$ be an encryption of the secret-key $\boldsymbol{s'} = \mathsf{Powersof2}(\boldsymbol{s}, k)$, where $\boldsymbol{q} \leftarrow (\mathbb{Z} \cap [0, 2^\gamma/p'))^{(k+1)\cdot\Theta}$ and $\boldsymbol{r} \leftarrow (\mathbb{Z} \cap (-2^\rho, 2^\rho))^{(k+1)\cdot\Theta}$. Let $\boldsymbol{c} = (\lfloor c \cdot y_i \rceil \bmod 2^{k+1})_{1 \le i \le \Theta}$ and let $\boldsymbol{c'} = \mathsf{BitDecomp}(\boldsymbol{c}, k)$ be the expanded ciphertext. Let $c'' = 2\langle \boldsymbol{\sigma}, \boldsymbol{c'} \rangle + [c]_2$. Then $c'' = q'' \cdot p' + r''$ where $r'' = \lfloor r \cdot p'/p \rceil + \delta'$ for some $\delta' \in \mathbb{Z}$ with $|\delta'| \le 2^{\rho+2} \cdot \Theta \cdot (k+1)$, and $[r]_2 = [r'']_2$.*

Proof. We have, from $\boldsymbol{\sigma} = p' \cdot \boldsymbol{q} + \boldsymbol{r} + \lfloor \boldsymbol{s'} \cdot p'/2^{k+1} \rceil$:

$$c'' = 2\langle \boldsymbol{\sigma}, \boldsymbol{c'} \rangle + [c]_2 = 2p' \cdot \langle \boldsymbol{q}, \boldsymbol{c'} \rangle + 2\langle \boldsymbol{r}, \boldsymbol{c'} \rangle + 2\left\langle \left\lfloor \boldsymbol{s'} \cdot \frac{p'}{2^{k+1}} \right\rceil, \boldsymbol{c'} \right\rangle + [c]_2 \tag{5}$$

Since the components of $\boldsymbol{c'}$ are bits, we have using $2\lfloor x/2 \rceil = x + \nu$ with $|\nu| \le 1$:

$$2\left\langle \left\lfloor \frac{p'}{2^{k+1}} \cdot \boldsymbol{s'} \right\rceil, \boldsymbol{c'} \right\rangle = \left\langle \frac{p'}{2^k} \cdot \boldsymbol{s'}, \boldsymbol{c'} \right\rangle + \nu_2 = \frac{p'}{2^k} \cdot \langle \boldsymbol{s'}, \boldsymbol{c'} \rangle + \nu_2$$

where $|\nu_2| \le \Theta \cdot (k+1)$. Using $\langle \boldsymbol{s'}, \boldsymbol{c'} \rangle = \langle \boldsymbol{s}, \boldsymbol{c} \rangle$ and since from Lemma 4 we have $\langle \boldsymbol{s}, \boldsymbol{c} \rangle = q' \cdot 2^k + r'$ with $[q']_2 = [q]_2$ and $r' = \lfloor r \cdot 2^k/p \rceil + \delta$ where $\delta \in \mathbb{Z}$ with $|\delta| \le \Theta/2 + 2$, we get:

$$2\left\langle \left\lfloor \frac{p'}{2^{k+1}} \cdot \boldsymbol{s'} \right\rceil, \boldsymbol{c'} \right\rangle = \frac{p'}{2^k} \cdot (q' \cdot 2^k + r') + \nu_2 = q' \cdot p' + \frac{p'}{2^k} \cdot r' + \nu_2 = q' \cdot p' + r \cdot \frac{p'}{p} + \nu_3$$

where $|\nu_3| \le |\nu_2| + \Theta/2 + 3 \le 2\Theta \cdot (k+1)$. Therefore we obtain from equation (5):

$$c'' = 2p' \cdot \langle \boldsymbol{q}, \boldsymbol{c'} \rangle + 2\langle \boldsymbol{r}, \boldsymbol{c'} \rangle + q' \cdot p' + r \cdot \frac{p'}{p} + \nu_3 + [c]_2 = q'' \cdot p' + r''$$

where $q'' := q' + 2\langle \boldsymbol{q}, \boldsymbol{c'} \rangle$ and $r'' = \lfloor r \cdot p'/p \rceil + \delta'$ for some $\delta' \in \mathbb{Z}$ with:

$$|\delta'| \le \left| 2\langle \boldsymbol{r}, \boldsymbol{c'} \rangle \right| + 1 + |\nu_3| + 1 \le 2^{\rho+1} \cdot \Theta \cdot (k+1) + 2\Theta \cdot (k+1) + 2 \le 2^{\rho+2} \cdot \Theta \cdot (k+1)$$

Eventually from $[c'']_2 = [c]_2$, $[c]_2 = [q]_2 \oplus [r]_2$, $[c'']_2 = [q'']_2 \oplus [r'']_2$ and $[q'']_2 = [q']_2 = [q]_2$, we obtain $[r]_2 = [r'']_2$ as required. □

5.3 The Modulus-Switching Algorithm for DGHV

From Lemma 5 we can now specify the modulus-switching algorithm for DGHV.

$\mathsf{SwitchKeyGen}(pk, sk, pk', sk')$:

1. Take as input two DGHV secret-keys p and p' of size η and η'. Let $\kappa = 2\gamma + \eta$ where γ is the size of the public key integers x_i under p.
2. Generate a vector $\boldsymbol{y}$ of Θ random numbers modulo $2^{\eta'+1}$ with κ bits of precision after the binary point, and a random vector $\boldsymbol{s}$ of Θ bits such that $2^{\eta'}/p = \langle \boldsymbol{s}, \boldsymbol{y} \rangle + \varepsilon \bmod 2^{\eta'+1}$ where $|\varepsilon| \le 2^{-\kappa}$. Generate the expanded secret-key $\boldsymbol{s'} = \mathsf{Powersof2}(\boldsymbol{s}, \eta')$

3. Compute a vector encryption $\boldsymbol{\sigma}$ of $\boldsymbol{s'}$ under sk', defined as follows:
$$\boldsymbol{\sigma} = p' \cdot \boldsymbol{q} + \boldsymbol{r} + \left\lfloor \boldsymbol{s'} \cdot \frac{p'}{2^{\eta'+1}} \right\rceil \tag{6}$$
where $\boldsymbol{q} \leftarrow (\mathbb{Z} \cap [0, q'_0))^{(\eta'+1)\cdot\Theta}$ and $\boldsymbol{r} \leftarrow (\mathbb{Z} \cap (-2^{\rho'}, 2^{\rho'}))^{(\eta'+1)\cdot\Theta}$, where q'_0 is from $x'_0 = q'_0 \cdot p' + r'$ in pk'.
4. Output $\tau_{pk\to pk'} = (\boldsymbol{y}, \boldsymbol{\sigma})$.

$\mathsf{SwitchKey}(\tau_{pk\to pk'}, c)$:

1. Let $\boldsymbol{y}, \boldsymbol{\sigma} \leftarrow \tau_{pk\to pk'}$
2. Compute the expanded ciphertext $\boldsymbol{c} = (\lfloor c \cdot y_i \rceil \bmod 2^{\eta'+1})_{1\le i\le\Theta}$ and let $\boldsymbol{c'} = \mathsf{BitDecomp}(\boldsymbol{c}, \eta')$.
3. Output $c'' = 2\langle \boldsymbol{\sigma}, \boldsymbol{c'} \rangle + [c]_2$.

5.4 The DGHV Scheme Without Bootstrapping

We are now ready to describe our DGHV variant in the BGV framework, that is without bootstrapping. As in [5] we construct a *leveled* fully homomorphic scheme, *i.e.* an encryption scheme whose parameters depend polynomially on the depth of the circuits that the scheme can evaluate.

Definition 2 (Leveled Fully Homomorphic Encryption [5]). *A family of homomorphic encryption schemes $\{\mathcal{E}^{(L)} : L \in \mathbb{Z}^+\}$ is said to be leveled fully homomorphic, if for all $L \in \mathbb{Z}^+$, $\mathcal{E}^{(L)}$ compactly evaluates all circuits of depth at most L, and the computational complexity of $\mathcal{E}^{(L)}$'s algorithms is polynomial (the same polynomial for all L) in the security parameter, L, and (for the evaluation algorithm) the size of the circuit.*

$\mathsf{FHE.KeyGen}(1^\lambda, 1^L)$. Take as input the security parameter λ and the number of levels L. Let μ be a parameter specified later. Generate a ladder of L decreasing moduli of size $\eta_i = (i+1)\mu$ from $\eta_L = (L+1)\mu$ down to $\eta_1 = 2\mu$. For each η_i run $\mathsf{DGHV.KeyGen}(1^\lambda)$ from Section 2 to generate a random odd integer p_i of size η_i; we take the same parameter γ for all i. Let pk_i be the corresponding public key and $sk_i = p_i$ be the corresponding secret-key. For $j = L$ down to 2 run $\tau_{pk_j\to pk_{j-1}} \leftarrow \mathsf{SwitchKeyGen}(pk_j, sk_j, pk_{j-1}, sk_{j-1})$. The full public key is $pk = (pk_L, \tau_{pk_L\to pk_{L-1}}, \ldots, \tau_{pk_2\to pk_1})$ and the secret-key is $sk = (p_1, \ldots, p_L)$.

$\mathsf{FHE.Encrypt}(pk, m \in \{0,1\})$. Run $\mathsf{DGHV.Encrypt}(pk_L, m)$.

$\mathsf{FHE.Decrypt}(sk, c)$. Suppose that the ciphertext is under modulus p_j. Output $m \leftarrow [c]_{p_j} \bmod 2$.

$\mathsf{FHE.Add}(pk, c_1, c_2)$. Suppose that the two ciphertexts c_1 and c_2 are encrypted under the same pk_j; if they are not, use $\mathsf{FHE.Refresh}$ below to make it so. First compute $c_3 \leftarrow c_1 + c_2$. Then output $c_4 \leftarrow \mathsf{FHE.Refresh}(\tau_{pk_j\to pk_{j-1}}, c_3)$, unless both ciphertexts are encrypted under pk_1; in this case, simply output c_3.

$\mathsf{FHE.Mult}(pk, c_1, c_2)$. Suppose that the two ciphertexts c_1 and c_2 are encrypted under the same pk_j; if they are not, use $\mathsf{FHE.Refresh}$ below to make it so. First compute $c_3 \leftarrow c_1 \cdot c_2$. Then output $c_4 \leftarrow \mathsf{FHE.Refresh}(\tau_{pk_j\to pk_{j-1}}, c_3)$, unless both ciphertexts are encrypted under pk_1; in this case, simply output c_3.

$\mathsf{FHE.Refresh}(\tau_{pk_{j+1}\to pk_j}, c)$. Output $c' \leftarrow \mathsf{SwitchKey}(\tau_{pk_{j+1}\to pk_j}, c)$.

Remark 4. As in [5] with the RLWE scheme, it is not really necessary to switch moduli after additions since additions increase the noise much more slowly than multiplications.

5.5 Correctness and Security

We show how to fix the parameter μ so that the ciphertext noise for every modulus in the ladder remains roughly the same, and we prove that FHE is a correct leveled FHE scheme.

Theorem 3. *For some $\mu = \mathcal{O}(\lambda + \log L)$, FHE is a correct L-leveled FHE scheme; specifically it correctly evaluates circuits of depth L with Add and Mult gates over $GF(2)$.*

Proof. First we show that for all levels j the size of the ciphertext c as input to $\mathsf{SwitchKey}$ satisfies the bound $|c| \leq 2^{\kappa_j}$ required from Lemma 5, where $\kappa_j = 2\gamma + \eta_j$. We distinguish two cases:

- Level $j = L$: a ciphertext generated using $\mathsf{FHE.Encrypt}$ has size at most γ. After the first multiplication by another ciphertext its size is then at most 2γ; therefore the previous bound is satisfied.
- Level $j < L$: the ciphertext $c'' = 2\langle\sigma, c'\rangle + [c]_2$ obtained from the $\mathsf{SwitchKey}$ algorithm from level $j+1$ has size at most $\gamma + \log_2(\Theta \cdot (\eta_{j+1} + 1))$. After multiplication by another ciphertext its size is then at most $2\gamma + 2\log_2(\Theta \cdot (\eta_{j+1} + 1))$; therefore the previous bound is also satisfied.

Now we show how to fix the parameter μ so that the ciphertext noise for every modulus in the ladder remains roughly the same. We say that a DGHV ciphertext c has noise r when $c = q \cdot p + r$ where $r = [c]_p$. As in [5] our strategy for setting the parameters is to pick a "universal" bound B on the ciphertext noise, and then prove that for all j a ciphertext under key pk_j has noise at most B; then it suffices to have $p_j > 2B^2$ for all j to ensure the correctness of the scheme.

Such a bound B certainly exists for fresh ciphertexts output by $\mathsf{FHE.Encrypt}$ under pk_L. Namely by correctly setting the parameters in $\mathsf{DGHV.Encrypt}$, their noise can be upper bounded by $2^{\rho'+2}$ for some $\rho' > \rho$. As in [5] the remainder of the proof is by induction: we will show that given two ciphertexts c_1 and c_2 under pk_j satisfying the bound B, the ciphertext $c' \leftarrow \mathsf{FHE.Mult}(pk, c_1, c_2)$ satisfies the bound for level $j-1$; the bound will also be satisfied for $\mathsf{FHE.Add}$ since ciphertext addition increases the noise much more slowly than multiplication.

The first step in $\mathsf{FHE.Mult}$ is an integer multiplication, which gives a ciphertext noise at most B^2. Then from Lemma 5 after modulus switching one gets a ciphertext c' with noise at most:

$$B^2 \cdot \frac{p_{j-1}}{p_j} + 2^{\rho+2} \cdot \Theta \cdot (\eta_{j-1} + 1)$$

If we choose B and our ladder of moduli p_i such that the two following properties hold:

- Property 1: $B \geq 2 \cdot 2^{\rho'+2} \cdot \Theta \cdot (\eta_{j-1} + 1)$ for all j.
- Property 2: $p_j / p_{j-1} \geq 2 \cdot B$ for all j.

then we obtain as required:

$$B^2 \cdot \frac{p_{j-1}}{p_j} + 2^{\rho'+2} \cdot \Theta \cdot (\eta_{j-1} + 1) \leq B^2 \cdot \frac{1}{2B} + \frac{B}{2} \leq B$$

From $\eta_{j-1} = j \cdot \mu$, Property 1 is satisfied if we take $B \geq 2^{\rho'+4} \cdot \Theta \cdot L \cdot \mu$. Moreover since p_j is a $(j+1)\mu$-bit integer for all j, we get $p_j/p_{j-1} \geq 2^{\mu-2}$. Therefore Property 2 is satisfied if $2^\mu \geq 8 \cdot B$. Therefore to satisfy both properties it suffices to select μ such that:

$$2^\mu \geq 2^{\rho'+7} \cdot \Theta \cdot L \cdot \mu$$

It suffices to take:

$$\mu = \rho' + 7 + 2\log_2 \rho' + 2\log_2 \Theta + 2\log_2 L$$

Since we have selected μ such that $2^{\mu} \geq 8 \cdot B$, we obtain for all levels $1 \leq j \leq L$:

$$p_j \geq 2^{\eta_j - 1} \geq 2^{(j+1)\cdot\mu - 1} \geq 2^{2\mu - 1} \geq 2 \cdot B^2$$

which ensures the scheme's correctness. Finally since $\rho' = \mathcal{O}(\lambda)$ and Θ is polynomial in λ, we have that $\mu = \mathcal{O}(\lambda + \log L)$ and the largest modulus in the system has size $\mathcal{O}(L \cdot (\lambda + \log L))$. This proves Theorem 3. □

We show that the resulting FHE is semantically secure under the following new assumption.

Definition 3 (Decisional Approximate GCD). *The (ρ, η, γ)-Decisional Approximate GCD Problem is: For a random η-bit odd integer p, given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$, and given an integer $z = x + b \cdot \lfloor 2^j \cdot p/2^{\eta+1} \rceil$ for a given random integer $j \in [0, \eta]$, where $x \leftarrow \mathcal{D}_{\gamma,\rho}(p)$ and $b \leftarrow \{0, 1\}$, find b.*

The Decisional Approximate GCD assumption is defined in the usual way. It is clearly stronger than the standard Approximate GCD assumption. We were not able to base the security of the leveled DGHV scheme on the standard Approximate GCD assumption; this is due to equation (6) which requires a non-standard encryption of the secret-key bits.

Theorem 4. FHE *is semantically secure under the Decisional Approximate GCD assumption and under the hardness of subset sum assumption.*

Proof. The FHE scheme is constructed from a sequence of DGHV schemes with the additional modulus-switching elements $\tau_{pk_j \to pk_{j-1}}$. A modulus-switching element $\tau_{pk \to pk'} = (\boldsymbol{y}, \boldsymbol{\sigma})$ contains a subset-sum sharing $\boldsymbol{y}$ of the secret $2^{\eta}/p$ corresponding to pk, and a "special" encryption $\boldsymbol{\sigma}$ under pk' of the secret-key sk corresponding to pk.

Therefore to prove the semantic security of the full FHE scheme it suffices to consider an enhanced scheme DGHV' with the subset-sum sharing $\boldsymbol{y}$ in the public key and with a special encrypt procedure $\mathsf{Encrypt}'$ corresponding to $\boldsymbol{\sigma}$. The semantic security of FHE then follows from the semantic security of DGHV' by using a standard hybrid argument.

$\mathsf{DGHV}'.\mathsf{KeyGen}(1^{\lambda})$. Run $\mathsf{DGHV}.\mathsf{KeyGen}(1^{\lambda})$ to generate a key-pair (pk, sk), and generate a subset-sum sharing $\boldsymbol{y}$ of $2^{\eta}/p$ as in the $\mathsf{SwitchKeyGen}$ procedure. The public-key is $pk' = (pk, \boldsymbol{y})$.

$\mathsf{DGHV}'.\mathsf{Encrypt}$ and $\mathsf{DGHV}'.\mathsf{Decrypt}$: same as DGHV.

$\mathsf{DGHV}'.\mathsf{Encrypt}'(sk, j, m)$: choose a random integer q in $[0, q_0)$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output the ciphertext:

$$c = m \cdot \left\lfloor 2^j \cdot \frac{p}{2^{\eta+1}} \right\rceil + r + q \cdot p$$

This completes the description of DGHV'. Note that we do not need to specify the decryption procedure corresponding to $\mathsf{Encrypt}'$. As in [9] the semantic security of DGHV with the additional element $\boldsymbol{y}$ follows from the security of the basic DGHV scheme recalled in Section 2 and from the hardness of subset-sum assumption, by using a hybrid argument[4].

[4] Note that the subset-sum need not be sparse.

We now consider the semantic security of DGHV' with $\mathsf{Encrypt}'$. Using a hybrid argument we can restrict ourselves to a fixed index $j \in [0, \eta]$. The output of $\mathsf{Encrypt}'(sk, j, m)$ is simulated by returning:

$$c = m \cdot z + r + 2 \sum_{i \in S} x_i \mod x_0$$

where $r \leftarrow \mathbb{Z} \cap (-2^{\rho'}, 2^{\rho'})$ and z is from the Decisional Approximate GCD instance. As in [9] one can show using the left-over hash lemma that this provides a statistically close simulation of $\mathsf{Encrypt}'(sk, j, m)$ when $b = 1$. With the same argument we have that the adversary gets no information about m when $b = 0$. There any non-negligible advantage in breaking the scheme's semantic security can be turned into an algorithm for breaking the Decisional Approximate GCD Assumption. This proves Theorem 4. □

6 Improved Attack against the Approximate GCD Algorithm

Recently, Chen and Nguyen [7] described an improved exponential algorithm for solving the approximate common divisor problem: they obtain a complexity of $\tilde{\mathcal{O}}(2^{\rho/2})$ for the partial version (with an exact multiple $x_0 = q_0 \cdot p$) and $\tilde{\mathcal{O}}(2^{3\rho/2})$ for the general version (with near-multiples only).[5]

In this section, we show that the latter complexity can be heuristically improved to $\tilde{\mathcal{O}}(2^{\rho})$ provided that sufficiently many near-multiples are available, which is the case in the DGHV scheme. Our algorithm has memory complexity $\tilde{\mathcal{O}}(2^{\rho})$, instead of only $\tilde{\mathcal{O}}(2^{\rho/2})$ for the Chen and Nguyen attack.

Indeed, assume that we have s large near-multiples $x_1, \ldots, x_s$ of a given prime p_0, of the hidden form $x_j = p_0 q_j + r_j$, where $q_j \in [0, 2^{\gamma}/p_0)$ (for γ polynomial in ρ) and $r_j \in [0, 2^{\rho})$ are chosen uniformly and independently at random. We claim that p_0 can then be recovered with overwhelming probability in time $\tilde{\mathcal{O}}(2^{\frac{s+1}{s-1}\rho})$ (and with significant probability in time $\tilde{\mathcal{O}}(2^{\frac{s}{s-1}\rho})$).

The algorithm is as follows. For $j = 1, \ldots, s$, let:

$$y_j = \prod_{i=0}^{2^{\rho}-1} (x_j - i)$$

Clearly, p_0 divides the GCD $g = \gcd(y_1, \ldots, y_s)$. Each y_i can be computed in time quasilinear in 2^{ρ} using a product tree, and the GCD can be evaluated as $\gcd(\cdots \gcd(\gcd(y_1, y_2), y_3), \ldots, y_s)$ using $s - 1$ quasilinear GCD computations on numbers of size $\mathcal{O}(2^{\rho} \cdot \gamma) = \tilde{\mathcal{O}}(2^{\rho})$. Hence, the whole computation of g takes time $\tilde{\mathcal{O}}(s \cdot 2^{\rho})$.

Now, we argue that with high probability on the choice of the (q_j, r_j), all the prime factors of g except p_0 are smaller than a bound B that is not much larger than 2^{ρ}. Then, p_0 can be recovered as g/g', where g' is the B-smooth part of g, which can in turn be computed in time quasilinear in $\max(B, |g|)$, *e.g.* using Bernstein's algorithm [2]. Overall, the full time complexity of the attack is thus $\tilde{\mathcal{O}}(\max(B, s \cdot 2^{\rho}))$, or simply $\tilde{\mathcal{O}}(B)$ assuming that $s = O(\rho)$, and without loss of generality that $B > 2^{\rho}$. All we need to find is how to choose B to obtain a sufficient success probability.

The probability that all the prime factors of g except p_0 are smaller than B is the probability that, for every prime $p \geq B$ other than p_0, not all the x_j's are congruent to one of $0, 1, \ldots, 2^{\rho} - 1 \bmod p$.

[5] Namely to solve the general version using the partial version algorithm it suffices to do exhaustive search on the ρ bits of noise in $x_0 = q_0 \cdot p + r_0$.

This happens with probability very close to $1 - (2^\rho/p)^s$. Hence, the probability that all the prime factors of g except p_0 are smaller than B is essentially given by the following Euler product:

$$P_{s,\rho}(B) = \prod_{\substack{p \geq B \\ p \neq p_0}} \left(1 - \frac{2^{s\rho}}{p^s}\right)$$

(which clearly converges to some positive value smaller than 1 since $s \geq 2$ and $B > 2^\rho$). We prove the following estimate on this Euler product.

Lemma 6. *For any $B > 2^{\rho+1/s}$, we have:*

$$1 - P_{s,\rho}(B) < \frac{2s}{s-1} \cdot \frac{2^{s\rho}}{B^{s-1}\log B}$$

Proof. Since $P_{s,\rho}(B) \in (0,1)$, we have:

$$1 - P_{s,\rho}(B) \leq -\log P_{s,\rho}(B) = \sum_{\substack{p \geq B \\ p \neq p_0}} \log\left(1 - \frac{2^{s\rho}}{p^s}\right)$$

Now for each $p \geq B$, we have $2^{s\rho}/p^s \leq 2^{s\rho}/B^s \leq 1/2$. Therefore, as $-\log x \leq 2\log 2 \cdot (1-x)$ for $x \in [1/2, 1]$, we get:

$$1 - P_{s,\rho}(B) \leq 2\log 2 \cdot 2^{s\rho} \sum_{p \geq B} \frac{1}{p^s}$$

Now observe that $1/p^s = s\int_p^{+\infty} dt/t^{s+1}$. Hence:

$$\sum_{p \geq B} \frac{1}{p^s} = s\int_B^{+\infty} \frac{\big(\pi(t) - \pi(B)\big)dt}{t^{s+1}} \leq 1.3s\int_B^{+\infty} \frac{dt}{t^s \log B} = \frac{1.3s}{(s-1)B^{s-1}\log B}$$

where π is the usual prime-counting function, which satisfies $\pi(x) < 1.3x/\log x$ (see e.g. [1, Th. 8.8.1]). Since $1.3\log 2 < 1$, this concludes the proof. □

In particular, if we pick $B = 2^{\frac{s}{s-1}\rho}$, we obtain $P_{s,\rho}(B) > 1 - 2/(\rho\log 2)$: thus, the problem can be solved in time $\tilde{\mathcal{O}}(2^{\frac{s}{s-1}\rho})$ with significant success probability. And if we pick $B = 2^{\frac{s+1}{s-1}\rho}$, we get $P_{s,\rho}(B) > 1 - 2^{-\rho}$: hence, the problem can be solved in time $\tilde{\mathcal{O}}(2^{\frac{s+1}{s-1}\rho})$ with an overwhelming success probability.

We see in both cases that for any given $\varepsilon > 0$, the complexity becomes $\mathcal{O}(2^{(1+\varepsilon)\rho})$ if s is large enough. Better yet, if $s = \omega(1)$ (for example $\Theta(\rho)$) near-multiples are available, the problem can be solved in time $\tilde{\mathcal{O}}(2^\rho)$ with overwhelming probability.

As in [7] we can perform a time-memory trade-off. First split the product y_1 into d sub-products z_k's, and guess which of these sub-products $z = z_k$ contains p_0. Let $g = \gcd(z, y_2, \ldots, y_s)$. The first GCD computation $\gcd(z, y_2)$ can be performed in time $\tilde{\mathcal{O}}(2^\rho)$ and memory $\tilde{\mathcal{O}}(2^\rho/d)$ by first computing $y_2 \bmod z$ using a product tree; the remaining gcd's can be computed with the same complexity; the same holds for recovering the B-smooth part of g. Hence p_0 can be recovered in time $\tilde{\mathcal{O}}(d \cdot 2^\rho)$ and memory $\tilde{\mathcal{O}}(2^\rho/d)$.

Instance	ρ	γ	$\log_2$ mem.	running time	running time [7]
Micro	12	10^4	26.3	40 s	
Toy (Section 8)	13	$61 \cdot 10^3$	29.9	13 min 22 s	
Toy' ([7] without x_0)	17	$1.6 \cdot 10^5$	35.3	17 h 50 min	*3495 hours*

Table 1. Running time of the attack, on a single core of an Amazon EC2 Cluster Compute Eight Extra Large Instance instance (featuring an Intel Xeon E5 processor at 2.5 GHz and 60.5 GB of memory), with parameter $s = \rho$. For the third instance, the running time of the Chen-Nguyen attack [7] was estimated by multiplying the running time from [7] (1.6 min) by 2^ρ.

6.1 Experimental Results

We have implemented the previous attack; see Appendix D for the source code. Table 1 shows that our attack performs well in practice; it is roughly 200 times faster than the corresponding attack of Chen and Nguyen for the smallest set of parameters considered in [7].

7 Implementation of DGHV with Compressed Public Key

In this section we describe an implementation of the DGHV scheme with the compression technique of Section 3; we use the variant with $x_0 = q_0 \cdot p$. However we don't use the quadratic encryption technique of [8] (as recalled and extended in Section 4); neither do we use the quadratic secret-key technique of [11] as both techniques are unnecessary for the full scheme when using compressed ciphertexts; this is because the "squashed decryption" procedure adds an incompressible additional term u_0 of size $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ to the public-key, so it is unnecessary to reduce the number of public-key elements x_i or the number of encrypted secret-key bits. As in [8] we use the optimization of [11] that splits the sparse secret-key s of dimension Θ into θ blocks of size $B = \Theta/\theta$ with a single non-zero bit each; see [11] for more details. We refer to Appendix A for a full description of the resulting scheme, and we provide the source code of our implementation in [18].

Asymptotic Key Size. To prevent lattice attacks against the sparse subset-sum problem, one must have $\Theta^2 = \gamma \cdot \omega(\log \lambda)$; see [8, 17] for more details. One can then take $\rho = \lambda$, $\eta = \tilde{\mathcal{O}}(\lambda^2)$, $\gamma = \tilde{\mathcal{O}}(\lambda^5)$, $\alpha = \tilde{\mathcal{O}}(\lambda^2)$, $\tau = \tilde{\mathcal{O}}(\lambda^3)$ and $\Theta = \tilde{\mathcal{O}}(\lambda^3)$. Using our compression technique the public key size is roughly $2\gamma + (\tau + \Theta) \cdot (\eta + \lambda) = \tilde{\mathcal{O}}(\lambda^5)$ bits.

Concrete Key Size and Execution Speed We have updated the parameters from [8] to take into account the improved approximate-GCD attack from [7]; see Table 2. The attack from [7] is memory bounded; however we took a conservative approach and considered a memory unbounded adversary. As in [8] we take $n = 4$ and $\theta = 15$ for all security levels. We can see in Table 2 that compression reduces the public key size considerably. In Appendix B we describe an optimization (not yet implemented) that further reduces the public key size by a factor 2. Table 3 shows no significant performance degradation with respect to [8].

8 Implementation of Leveled DGHV

In this section we describe an implementation of the leveled DGHV scheme described in Section 5 in the BGV framework. We implement the modulus-switching procedure as described in Section

Instance	λ	ρ	η	$\gamma \times 10^{-6}$	α	τ	Θ	pk size
Toy	42	27	1026	0.15	936	158	144	77 KB
Small	52	41	1558	0.83	1476	572	533	437 KB
Medium	62	56	2128	4.20	2016	2110	1972	2207 KB
Large	72	71	2698	19.35	2556	7659	7897	10.3 MB

Table 2. The concrete parameters of various test instances and their respective public key sizes, for DGHV with compressed public-key.

Instance	KeyGen	Encrypt	Decrypt	Expand	Recrypt
Toy	0.06 s	0.05 s	0.00 s	0.01 s	0.41 s
Small	1.3 s	1.0 s	0.00 s	0.15 s	4.5 s
Medium	28 s	21 s	0.01 s	2.7 s	51 s
Large	10 min	7 min 15 s	0.05 s	51 s	11 min 34 s

Table 3. Timings of our Sage 4.7.2 [16] code (single core of a desktop computer with an Intel Core2 Duo E8400 at 3 GHz), for DGHV with compressed public-key.

5.3, with an optimization of the ciphertext expansion procedure (see below). We also implement the bootstrapping operation; although not strictly necessary, this enables to get a FHE that can perform homomorphic evaluations indefinitely without needing to specify at setup time a bound on the multiplicative level. It is also interesting to compare the running time of the bootstrapping operation between the non-leveled DGHV scheme of Section 7 and the leveled DGHV scheme.

8.1 Faster Ciphertext Expansion

We consider the modulus-switching procedure of Section 5.3. The initial modulus p has size η and the new modulus p' has size $\eta' < \eta$. The first modulus p is shared among the y_i elements as

$$\frac{2^{\eta'}}{p} = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon \mod 2^{\eta'+1} \tag{7}$$

where the s_i's are bits, the y_i's have κ bits of precision after the binary point, and $|\varepsilon| \leq 2^{-\kappa}$. In practice one can generate the y_i's pseudo-randomly (except y_1), as in Appendix A. However the ciphertext expansion from Step 2 of $\mathsf{SwitchKey}$ algorithm (Section 5.3) is a time-consuming procedure.

Therefore instead of using pseudo-random y_i's we use the following (admittedly aggressive) optimization. Let δ be a parameter specified later. We generate a random y with $\kappa + \delta \cdot \Theta \cdot \eta$ bits of precision after the binary point, and we define the y_i's for $2 \leq i \leq \Theta$ as:

$$y_i = \left[y \cdot 2^{i \cdot \delta \cdot \eta} \right]_{2^{\eta'+1}}$$

keeping only κ bits of precision after the binary point for each y_i as previously. We fix y_1 so that equality (7) holds, assuming $s_1 = 1$. Then the ciphertext expansion from Step 2 of the $\mathsf{SwitchKey}$ algorithm (Section 5.3) can be computed as follows, for all $2 \leq i \leq \Theta$:

$$z_i = \lfloor c \cdot y_i \rceil \bmod 2^{\eta'+1} = \lfloor c \cdot y \cdot 2^{i \cdot \delta \cdot \eta} \rceil \bmod 2^{\eta'+1}$$

Therefore computing all the z_i's (except z_1) is now essentially a single multiplication $c \cdot y$. In Appendix C we describe a lattice attack against this optimization; we show that the attack is thwarted by selecting δ such that $\delta \cdot \Theta \cdot \eta \geq 3\gamma$.

Finally we use the following straightforward optimization: instead of using BitDecomp and Powersof2 with bits, we use words of size ω bits instead. This decreases the running time of SwitchKey by a factor of about ω, at the cost of increasing the resulting noise by roughly ω bits. We took $\omega = 32$ in our implementation.

8.2 Bootstrapping: The Decryption Circuit.

Recall that the decryption function in the DGHV scheme is:

$$m \leftarrow \left[c - \left\lfloor \sum_{i=1}^{\Theta} s_i \cdot z_i \right\rceil \right]_2 \tag{8}$$

where $z_i = [c \cdot y_i]_2$ for $1 \leq i \leq \Theta$ is the expanded ciphertext, keeping only $n = \lceil \log_2(\theta + 1) \rceil$ bits of precision after the binary point for each z_i. The s_i's form a sparse Θ-dimensional vector of Hamming weight θ, such that:

$$\frac{1}{p} = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

where the y_i's have κ bits of precision after the binary point, and $|\varepsilon| \leq 2^{-\kappa}$. Note that for bootstrapping the decryption circuit is only used for the smallest modulus p in the ladder. The following lemma shows that the message m can be computed using a circuit of multiplicative depth exactly n.

Lemma 7. *Let $a = [a_0, \ldots, a_n]$ and $b = [b_0, \ldots, b_n]$ be two integers of size $n+1$ bits, where every bit a_i and b_i has multiplicative depth at most i. Then every bit c_i of the sum $c = (a+b) \bmod 2^{n+1} = [c_0, \ldots, c_n]$ has multiplicative depth at most i.*

Proof. Let δ_i be the i-th carry bit, with $\delta_0 = 0$. We have $c_i = a_i \oplus b_i \oplus \delta_i$ for $0 \leq i \leq n$, where $\delta_i = a_{i-1} \cdot b_{i-1} + a_{i-1} \cdot \delta_{i-1} + b_{i-1} \cdot \delta_{i-1}$ for $1 \leq i \leq n$. Therefore by recursion δ_i has multiplicative depth at most i; this implies that c_i has multiplicative depth at most i. □

Therefore using a simple loop the sum of the Θ numbers $s_i \cdot z_i$ in equation (8) can be computed with a circuit of multiplicative depth n. Since a subsequent homomorphic operation (either addition or multiplication) must be possible between refreshed ciphertexts, the full bootstrapping procedure requires a leveled FHE scheme with multiplicative depth $L = n + 1$. Note that for bootstrapping an encryption of the secret-key bits s_i (corresponding to the last modulus p_1 in the ladder) must be provided under p_L, the first modulus in the ladder, so that the homomorphic evaluation of m in equation (8) can start under the public key pk_L.

As in Section 7 we use the optimization from [11] that splits the sparse secret-key s into θ blocks of size $B = \Theta/\theta$ with a single non-zero bit each, so that the sum of Θ elements can be split into θ sub-sums of B elements.

8.3 Implementation Results

In this section we describe an implementation of the leveled DGHV scheme, including the bootstrapping operation. As mentioned previously we cannot use the variant with noise-free $x_0 = q_0 \cdot p$ since otherwise p could be recovered using the ECM; namely the smallest modulus in the ladder has size only $2\mu = 164$ bits for the "Large" instance.

Asymptotic key size. Using $\theta = \lambda$ the degree of the decryption polynomial is $\mathcal{O}(\lambda)$ and the multiplicative depth is $L = \mathcal{O}(\log \lambda)$. This gives $\mu = \mathcal{O}(\lambda + \log L) = \mathcal{O}(\lambda)$ and the largest modulus has size $\eta = \mathcal{O}(L \cdot (\lambda + \log L)) = \tilde{\mathcal{O}}(\lambda)$. For the constraint $\gamma = \omega(\eta^2 \cdot \log \lambda)$ we can take $\gamma = \tilde{\mathcal{O}}(\lambda^3)$ and for the constraint $\Theta^2 = \gamma \cdot \omega(\log \lambda)$ we can take $\Theta = \tilde{\mathcal{O}}(\lambda^2)$. From the constraint $\alpha \cdot \tau \geq \gamma + \omega(\log \lambda)$ we can take $\alpha = \tilde{\mathcal{O}}(\lambda)$ and $\tau = \tilde{\mathcal{O}}(\lambda^2)$. The public-key size with compressed ciphertexts is then approximately $L \cdot (\gamma + \tau \cdot (\eta + \lambda) + \Theta \cdot (\eta + \lambda)) = \tilde{\mathcal{O}}(\lambda^3)$.

Concrete Key Size and Execution Speed We summarize in Tables 4 and 5 the performance of our implementation of the leveled DGHV scheme. We denote by η the size of the largest modulus in the ladder. The moduli have size $\eta_i = (i + 1) \cdot \mu$ bits for $1 \leq i \leq L$. For simplicity we have used the same value of Θ and γ for all levels in the ladder. The running time of the Recrypt operation is disappointing compared to the non-leveled implementation from Section 7; however we think that there is room for improvement.

Instance	λ	ρ	η	μ	$\gamma \times 10^{-6}$	Θ	pk size
Toy	42	14	336	56	0.061	195	354 KB
Small	52	20	390	65	0.27	735	1690 KB
Medium	62	26	438	73	1.02	2925	7.9 MB
Large	72	34	492	82	2.20	5700	18 MB

Table 4. The concrete parameters of various test instances and their respective public-key sizes for leveled DGHV.

Instance	KeyGen	Encrypt	Decrypt	Mult & Scale	Recrypt
Toy	0.36 s	0.01 s	0.00 s	0.04 s	8.8 s
Small	5.4 s	0.07 s	0.00 s	0.59 s	101 s
Medium	1 min 12 s	0.85 s	0.00 s	9.1 s	32 min 38 s
Large	6 min 18 s	3.4 s	0.00 s	41 s	2 h 27 min

Table 5. Timings of our Sage 4.7.2 [16] code (single core of a desktop computer with an Intel Core2 Duo E8400 at 3 GHz).

Acknowledgments

We would like to thank Tancrède Lepoint, Phong Nguyen and the EUROCRYPT referees for their helpful comments.

References

1. E. Bach and J. Shallit, *Algorithmic Number Theory*, vol. 1, MIT Press, 1996.
2. D.J. Bernstein, *How to Find Smooth Parts of Integers*, 2004. Available at `http://cr.yp.to/papers.html#smoothparts`.

3. Z. Brakerski and V. Vaikuntanathan, *Efficient Fully Homomorphic Encryption from (Standard) LWE*. Proceedings of FOCS 2011. Full version available at IACR eprint.
4. Z. Brakerski and V. Vaikuntanathan, *Fully Homomorphic Encryption for Ring-LWE and Security for Key Dependent Messages*. In P. Rogaway (Ed.), *CRYPTO 2011*, LNCS, vol. 6841, Springer, 2011, pp. 505–524.
5. Z. Brakerski, C. Gentry and V. Vaikuntanathan, *Fully Homomorphic Encryption without Bootstrapping*. Cryptology ePrint Archive, Report 2011/277.
6. A. Cafure and G. Matera, *Improved explicit estimates on the number of solutions of equations over a finite field*. Finite Fields and Their Applications, vol. 12(2), 2006, pp. 155-185.
7. Y. Chen and P.Q. Nguyen, *Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers*. Cryptology ePrint Archive, Report 2011/436.
8. J.S. Coron, A. Mandal, D. Naccache and M. Tibouchi, *Fully Homomorphic Encryption over the Integers with Shorter Public Keys*. In P. Rogaway (Ed.), *CRYPTO 2011*, LNCS, vol. 6841, Springer, 2011, pp. 487–504. Full version available at IACR eprint.
9. M. van Dijk, C. Gentry, S. Halevi and V. Vaikuntanathan, *Fully homomorphic encryption over the integers*. In H. Gilbert (Ed.), *EUROCRYPT 2010*, LNCS, vol. 6110, Springer, 2010, pp. 24–43.
10. C. Gentry, *A fully homomorphic encryption scheme*. Ph.D. thesis, Stanford University, 2009. Available at `http://crypto.stanford.edu/craig`.
11. C. Gentry and S. Halevi, *Implementing Gentry's fully homomorphic encryption scheme*. In K. Paterson (Ed.), *EUROCRYPT 2011*, LNCS, vol. 6632, Springer, 2011, pp. 129–148.
12. K. Lauter, M. Naehrig and V. Vaikuntanathan, *Can Homomorphic Encryption be Practical?*, Cryptology ePrint Archive, Report 2011/405.
13. A. K. Lenstra, *Generating RSA Moduli with a Predetermined Portion*. In K. Ohta and D. Pei (Eds.), *ASIACRYPT 1998*, LNCS, vol. 1514, Springer, 1998, pp. 1–10.
14. H.W. Lenstra, *Factoring integers with elliptic curves*. Annals of Mathematics, vol. 126(3): 1987, pp. 649–673.
15. N.P. Smart and F. Vercauteren, *Fully homomorphic encryption with relatively small key and ciphertext sizes*. In P.Q. Nguyen and D. Pointcheval (Eds.), *PKC 2010*, LNCS, vol. 6056, Springer, 2010, pp. 420–443.
16. W.A. Stein et al., *Sage Mathematics Software* (Version 4.7.2), The Sage Development Team, 2011, `http://www.sagemath.org`.
17. D. Stehlé and R. Steinfeld, *Faster fully homomorphic encryption*. In M. Abe (Ed.), *ASIACRYPT 2010*, LNCS, vol. 6477, Springer, 2010, pp. 377–394.
18. `https://github.com/coron/fhe`

A Complete Description of the DGHV Variant with Compressed Public Key

We provide a complete description of the FHE with the ciphertext compression technique. Note that the ciphertext compression technique is applied to both the public key elements x_i of the somewhat homomorphic scheme and to the encryption σ_i of the secret-key bits.

$\mathsf{KeyGen}(1^\lambda)$. Generate a random prime integer p of size η bits. Pick a random odd integer $q_0 \in [0, 2^\gamma/p)$ and let $x_0 = q_0 \cdot p$. Initialize a pseudo-random number generator f_1 with a random seed se_1. Use $f_1(\mathsf{se}_1)$ to generate a set of integers $\chi_i \in [0, 2^\gamma)$ for $1 \le i \le \tau$. For all $1 \le i \le \tau$ compute:

$$\delta_i = \langle \chi_i \rangle_p + \xi_i \cdot p - r_i$$

where $r_i \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ and $\xi_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda+\eta}/p)$. Let $pk^* = (\mathsf{se}_1, x_0, \delta_1, \ldots, \delta_\tau)$. The corresponding integers x_i for $1 \le i \le \tau$ are defined as $x_i = \chi_i - \delta_i$.

Additionally generate a random bit vector $\boldsymbol{s}$ of length Θ, subject to the conditions that $s_1 = 1$, that for each $k \in [0, \theta)$, there is at most one nonzero bit among the s_i's for $k \cdot B + 1 \le i < (k+1) \cdot B + 1$, where $B = \lfloor \Theta/\theta \rfloor$, and that the Hamming weight of $\boldsymbol{s}$ is exactly θ.

Initialize a pseudo-random number generator f_2 with a random seed se_2, and use $f(\mathsf{se}_2)$ to generate integers $u_i \in [0, 2^{\kappa+1})$ for $2 \le i \le \Theta$, where $\kappa := \gamma + n + 2$. Then, set u_1 such that:

$$\sum_{i=1}^{\Theta} s_i \cdot u_i = x_p \mod 2^{\kappa+1}$$

where $x_p \leftarrow \lfloor 2^\kappa / p \rceil$.

Initialize a pseudo-random number generator f_3 with a random seed se_3, and compute encryptions $\boldsymbol{\sigma}$ of the vector $\boldsymbol{s}$ as follows: use $f_3(\mathsf{se}_3)$ to generate integers $\chi'_i \in [0, 2^\gamma[$ for every $1 \le i \le \Theta$; generate random integers $r'_i \in (-2^\rho, 2^\rho)$ and $\xi'_i \in [0, 2^{\gamma+\eta}/p)$, and let:

$$\delta'_i = \langle \chi'_i \rangle_p + \xi'_i \cdot p - 2 \cdot r'_i - s_i$$

The corresponding encryption σ_i of s_i is then defined as:

$$\sigma_i = \chi'_i - \delta'_i$$

Finally, output the secret key $sk = \boldsymbol{s}$ and the public key $pk = (pk^*, \mathsf{se}_2, u_1, \mathsf{se}_3, \boldsymbol{\delta'})$.

$\mathsf{Encrypt}(pk, m \in \{0,1\})$. Recover the integers x_i from pk^*. Choose a random integer vector $\boldsymbol{b} = (b_i)_{1 \le i \le \tau} \in [0, 2^\alpha)^\tau$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$. Output the ciphertext:

$$c^* = m + 2r + 2 \sum_{i=1}^{\tau} b_i \cdot x_i \mod x_0$$

$\mathsf{Add}(pk, c_1^*, c_2^*)$. Output $c_1^* + c_2^* \bmod x_0$.

$\mathsf{Mult}(pk, c_1^*, c_2^*)$. Output $c_1^* \cdot c_2^* \bmod x_0$.

$\mathsf{Expand}(pk, c^*)$. The ciphertext expand procedure takes a ciphertext c^* and computes the associated expanded ciphertext. To do so, for every $1 \le i \le \Theta$ first compute the random integer u_i's using the seeded pseudo-random number generator $f_2(\mathsf{se}_2)$, then let $y_i = u_i / 2^\kappa$ and compute z_i given by:

$$z_i = [c^* \cdot y_i]_2$$

keeping only $n = \lceil \log_2(\theta + 1) \rceil$ bits of precision after the binary point. Define the vector $\boldsymbol{z} = (z_i)$. Output the expanded ciphertext $c = (c^*, \boldsymbol{z})$.

$\mathsf{Decrypt}(sk, c^*, \boldsymbol{z})$. Output $m \leftarrow \left[c^* - \lfloor \sum_{i=1}^{\Theta} s_i \cdot z_i \rceil \right]_2$.

$\mathsf{Recrypt}(pk, c^*, \boldsymbol{z})$. Recover the encrypted secret-key bits σ_i from pk. Apply the decryption circuit to the expanded ciphertext $\boldsymbol{z}$ and the encrypted secret key bits σ_i. Output the result as a refreshed ciphertext c^*_{new}.

This completes the description of the scheme.

B Optimization with Ciphertext Pairs

We briefly describe an optimization (not yet implemented) that further reduces the public key size by a factor 2. We first generate a random integer Γ modulo p that must remain secret. Then to encrypt a secret-key bit s for bootstrapping instead of a single ciphertext we generate a *pair* of ciphertexts:

$$\sigma_0 = q_0 \cdot p + \Gamma \cdot r_0 \quad \text{and} \quad \sigma_1 = q_1 \cdot p + (\Gamma^{-1} \bmod p) \cdot r_1$$

for randoms r_0, r_1 of size $\rho/2$ bits (instead of ρ bits), with $s = [r_0]_2 = [r_1]_2$. Then during recryption whenever a product $s \cdot s'$ must be evaluated homomorphically, we compute the ciphertext product

$\sigma_0 \cdot \sigma'_1$ or $\sigma_1 \cdot \sigma'_0$, where (σ'_0, σ'_1) is a ciphertext pair corresponding to s', that is $\sigma'_0 = q'_0 \cdot p + \mathit{\Gamma} \cdot r'_0$ and $\sigma'_1 = q'_1 \cdot p + (\mathit{\Gamma}^{-1} \bmod p) \cdot r'_1$. We obtain that the factor $\mathit{\Gamma}$ cancels in the product:

$$\sigma_0 \cdot \sigma'_1 = q'' \cdot p + r_0 \cdot r'_1$$

so a ciphertext of noise size ρ (instead of 2ρ) is obtained. Therefore with the same decryption polynomial to be evaluated homomorphically, the size η of p can be divided by 2, and from the asymptotic condition $\gamma = \eta^2 \cdot \omega(\log \lambda)$ the ciphertext size γ is divided by 4. However twice more ciphertexts are required for encrypting the secret-key bits, so asymptotically the public-key size is divided by 2.

Heuristically it is safe to have ciphertexts (σ_0, σ_1) with only $\rho/2$-bit noise, since the randoms r_0, r_1 are further "masked" with the secret integer $\mathit{\Gamma}$ modulo p.

C Lattice Attack against the Ciphertext Expansion Optimization

In this section we describe a lattice attack against the optimization of ciphertext expansion described in Section 8.1; we show that the attack is thwarted by selecting $\delta \geq 3$. We consider the equation (7) from Section 8.1:

$$\frac{2^{\eta'}}{p} = \sum_{i=1}^{\mathit{\Theta}} s_i \cdot y_i + \varepsilon \mod 2^{\eta'+1}$$

where the s_i's are bits and $|\varepsilon| \leq 2^{-\kappa}$, and for $2 \leq i \leq \mathit{\Theta}$ we have:

$$y_i = \left[y \cdot 2^{i \cdot \delta \cdot \eta} \right]_{2^{\eta'+1}}$$

with κ bits of precision after the binary point, where y is a random number with $\ell := \kappa + \delta \cdot \mathit{\Theta} \cdot \eta$ bits of precision after the binary point. We obtain, assuming that $s_1 = 1$:

$$\frac{2^{\eta'}}{p} = y_1 + \sum_{i=2}^{\mathit{\Theta}} s_i \cdot y \cdot 2^{\delta \cdot i \cdot \eta} + \varepsilon \mod 2^{\eta'+1}$$

Letting:

$$S := \sum_{i=2}^{\mathit{\Theta}} s_i \cdot 2^{\delta \cdot i \cdot \eta}$$

we obtain:

$$\frac{2^{\eta'}}{p} = y_1 + S \cdot y + \varepsilon \mod 2^{\eta'+1}$$

This gives:

$$2^{\eta'} = p \cdot y_1 + S \cdot p \cdot y + \varepsilon \cdot p + \mathit{\Delta} \cdot 2^{\eta'+1}$$

for some $\mathit{\Delta} \in \mathbb{Z}$. Let $Y_1 := 2^{\ell} \cdot y_1$ and $Y := 2^{\ell} \cdot y$ which gives $Y_1, Y \in \mathbb{Z}$; we obtain:

$$2^{\eta'+\ell} = p \cdot Y_1 + S \cdot p \cdot Y + u \mod 2^{\ell+\eta'+1}$$

where all variables are now over $\mathbb{Z}$, with $|u| \leq |2^{\ell} \cdot \epsilon \cdot p| \leq 2^{\ell-\kappa+\eta}$. The previous equation can be linearized by letting $x_1 := p$, $x_2 := p \cdot S$ and $x_3 := u$, which gives:

$$2^{\eta'+\ell} = x_1 \cdot Y_1 + x_2 \cdot Y + x_3 \mod 2^{\ell+\eta'+1} \tag{9}$$

where the three unknowns x_1, x_2 and x_3 are small. Using LLL equation (9) can be solved if the product of the three unknowns is less than the modulus, which gives the condition:

$$\eta + (\eta + \delta \cdot \Theta \cdot \eta) + (\ell - \kappa + \eta) \leq \ell + \eta' + 1$$

which gives:

$$3\eta + \delta \cdot \Theta \cdot \eta \leq \kappa + \eta' + 1$$

which using $\eta' + 1 \leq \eta$ gives the necessary condition:

$$2\eta + \delta \cdot \Theta \cdot \eta \leq \kappa$$

Using $\kappa = 2\gamma + \eta$ from Section 5.3, we get:

$$\eta + \delta \cdot \Theta \cdot \eta \leq 2\gamma$$

which gives the necessary condition:

$$\delta \cdot \Theta \cdot \eta \leq 2\gamma$$

To prevent the attack we select δ such that $\delta \cdot \Theta \cdot \eta \geq 3\gamma$.

D Source Code of the GACD Attack

```
def genXi(rho,eta,gam,p):
  return p*ZZ.random_element(2^(gam-eta))+ZZ.random_element(2^rho)

def attackGACD(rho=12,gam=1000,eta=100):
  p=random_prime(2^eta)
  print "p=",p

  t=cputime(subprocesses=True)
  s=rho

  B=floor(2^(1.*rho*(s+1)/(s-1)))
  fa=factorial(B)

  for j in range(1,s):
    x=genXi(rho,eta,gam,p)
    z=prod([x-i for i in range(2^rho)])
    if j==1:
      g=z
      continue

    g=prime_to_m_part(gcd(g,z),fa)

    print "j=",j,"gcd size=",g.nbits()
    if g.nbits()==p.nbits(): break

  print g==p,cputime(t)
```