Batch Fully Homomorphic Encryption over the Integers*

Jung Hee Cheon[1], Jean-Sébastien Coron[2], Jinsu Kim[1], Moon Sung Lee[1], Tancrède Lepoint[3,4], Mehdi Tibouchi[5], and Aaram Yun[6]

[1] Seoul National University (SNU), Republic of Korea
{jhcheon,kjs2002,moolee}@snu.ac.kr
[2] Tranef, France
jscoron@tranef.com
[3] CryptoExperts, France
[4] École Normale Supérieure, France
tancrede.lepoint@cryptoexperts.com
[5] NTT Secure Platform Laboratories, Japan
tibouchi.mehdi@lab.ntt.co.jp
[6] Ulsan National Institute of Science and Technology (UNIST), Republic of Korea
aaramyun@unist.ac.kr

Abstract. We extend the fully homomorphic encryption scheme over the integers of van Dijk *et al.* (DGHV) into a batch fully homomorphic encryption scheme, *i.e.* to a scheme that supports encrypting and homomorphically processing a vector of plaintexts as a single ciphertext. We present two variants in which the semantic security is based on different assumptions. The first variant is based on a new decisional problem, the Decisional Approximate-GCD problem, whereas the second variant is based on the more classical computational Error-Free Approximate-GCD problem but requires additional public key elements.
We also show how to perform arbitrary permutations on the underlying plaintext vector given the ciphertext and the public key. Our scheme offers competitive performance even with the bootstrapping procedure: we describe an implementation of the homomorphic evaluation of AES, with an amortized cost of about 12 minutes per AES ciphertext on a standard desktop computer; this is comparable to the timings presented by Gentry *et al.* at Crypto 2012 for their implementation of a Ring-LWE based fully homomorphic encryption scheme.

Keywords: Fully Homomorphic Encryption, Batch Encryption, Chinese Remainder Theorem, Approximate GCD, Homomorphic AES.

1 Introduction

Fully Homomorphic Encryption (FHE). Fully homomorphic encryption allows a worker to perform implicit additions and multiplications on plaintext

* This paper is a merger of two independent works [CLT13,KLYC13] built on the same basic idea but with different contributions. The respective full versions are posted on ePrint.

values while exclusively manipulating encrypted data. The first construction of a fully homomorphic scheme (based on ideal lattices) was described by Gentry in [Gen09], and proceeds in several steps. First, one constructs a *somewhat homomorphic* encryption scheme, which only supports a limited number of multiplications: ciphertexts contain some noise that becomes larger with successive homomorphic multiplications, and only ciphertexts whose noise size remains below a certain threshold can be decrypted correctly. The second step is to *squash* the decryption procedure associated with an arbitrary ciphertext so that it can be expressed as a low degree polynomial in the secret key bits. Then, Gentry's key idea, called *bootstrapping*, consists in homomorphically evaluating this decryption polynomial on encryptions of the secret key bits, resulting in a different ciphertext associated with the same plaintext, but with possibly reduced noise. This *refreshed* ciphertext can then be used in subsequent homomorphic operations. By repeatedly refreshing ciphertexts, the number of homomorphic operations becomes unlimited, resulting in a fully homomorphic encryption scheme.

Since Gentry's breakthrough result, many improvements have been made, introducing new variants, improving efficiency, and providing new features. Recently, Brakerski, Gentry and Vaikuntanathan described a different framework where the ciphertext noise grows only linearly with the multiplicative level instead of exponentially [BGV12], so that bootstrapping is no longer necessary to obtain a scheme supporting the homomorphic evaluation of any given polynomial size circuit. Currently three main families of fully homomorphic encryption schemes are known:

1. Gentry's original scheme [Gen09] based on ideal lattices. An implementation of Gentry's scheme was proposed by Gentry and Halevi in [GH11] with a public key of 2.3 GB and a ciphertext refresh procedure of 30 minutes; the implementation is based on many interesting algorithmic optimizations, including some borrowed from Smart and Vercauteren [SV10].
2. van Dijk, Gentry, Halevi and Vaikuntanathan's (DGHV) scheme over the integers [DGHV10]. It was recently shown how to significantly reduce the public key size in DGHV, yielding a 10.3 MB public key and an 11-minute refresh procedure [CNT12].
3. Brakerski and Vaikuntanathan's scheme based on the Learning with Errors (LWE) and Ring Learning with Errors (RLWE) problems [BV11a,BV11b], and follow-up works (*e.g.* the scale-free variant of Brakerski [Bra12] and the NTRU-variant [LATV12]). An implementation is described in [GHS12b] with an efficient (given the current state of knowledge) homomorphic evaluation of the full AES encryption circuit. The authors use the *batch* RLWE-based scheme proposed in [BGV12,GHS12a], that allows one to encrypt vectors of plaintexts in a single ciphertext and to perform any permutation on the underlying plaintext vector while manipulating only the ciphertext [SV11].

Our Contributions. In this paper we focus on the DGHV scheme. Our goal is to extend DGHV to support the same batching capability as in RLWE-based

schemes [BGV12,GHS12a], and to homomorphically evaluate a full AES circuit with roughly the same level of efficiency as [GHS12b], in order to obtain an implementation of a FHE scheme with similar features but based on different techniques and assumptions.

In the original DGHV scheme, a ciphertext has the form

$$c = q \cdot p + 2r + m$$

where p is the secret key, q is a large random integer, and r is a small random integer (noise); the bit message $m \in \{0, 1\}$ is recovered by computing $m = [c \bmod p] \bmod 2$. The scheme is clearly homomorphic for both addition and multiplication, since addition and multiplication of ciphertexts correspond to addition and multiplication of plaintexts modulo 2.

To encrypt multiple bits m_i into a single ciphertext c, we use the Chinese Remainder Theorem with respect to a tuple of $(\ell+1)$ coprime integers $q_0, p_0, \ldots, p_{\ell-1}$. The batch ciphertext has the form

$$c = \mathsf{CRT}_{q_0,p_0,\ldots,p_{\ell-1}}(q, 2r_0 + m_0, \ldots, 2r_{\ell-1} + m_{\ell-1}),$$

and correctly decrypts to the bit vector (m_i) given by $m_i = [c \bmod p_i] \bmod 2$ for all $0 \leqslant i < \ell$.[7] Modulo each of the p_i's the ciphertext c behaves as in the original DGHV scheme. Accordingly, the addition or multiplication of two ciphertexts yields a new ciphertext that decrypts to the componentwise sum or product modulo 2 of the original plaintexts.

The main challenge, however, was to prove the semantic security of our new scheme. In the original DGHV scheme, public-key encryption is performed by masking the message m with a random subset sum of the public key elements $x_j = q_j \cdot p + r_j$ as

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

The semantic security is proved by applying the Leftover Hash Lemma on the subset sum modulo q_0, and using the random $2r$ in (1) to further randomize the ciphertext modulo p.

To prove semantic security for the batch scheme our first technique is to rely on a new assumption, namely the Decisional Approximate-GCD assumption. Under this assumption the integers x_j in the subset-sum from (1) are assumed to be indistinguishable from random modulo x_0; semantic security is then proved by applying the Leftover Hash Lemma modulo x_0; the additional random $2r$ in (1) becomes unnecessary. Extending DGHV public-key encryption to the batch setting is then straightforward; namely one can use the same random subset sum technique with public key elements x_j having a small residue modulo each of the p_i's instead of only modulo p. We show that our batch DGHV scheme can encrypt $\ell = \tilde{\mathcal{O}}(\lambda^3)$ bits in a single ciphertext; therefore the ciphertext expansion ratio becomes $\tilde{\mathcal{O}}(\lambda^2)$ instead of $\tilde{\mathcal{O}}(\lambda^5)$ in the original scheme.

[7] We denote by $\mathsf{CRT}_{q_0,p_0,\ldots,p_{\ell-1}}(q, a_0, \ldots, a_{\ell-1})$ the unique integer u with $0 \leqslant u < q_0 \cdot \prod_{i=0}^{\ell-1} p_i$ such that $u \equiv q \pmod{q_0}$ and $u \equiv a_i \pmod{p_i}$ for all $0 \leqslant i < \ell$.

In Section 4 we describe a different technique that does not rely on a new decisional assumption but on the known (computational) Error-Free Approximate-GCD assumption used in previous work [DGHV10,CMNT11,CNT12]. For the proof of semantic security to go through in the batch setting, the ciphertext c should be independently randomized modulo each of the p_i's, which is not easy to achieve without knowing the p_i's. Indeed, if we only use a single additive term $2r$ as in Equation (1), then the *same* random term $2r = 2r \bmod p_i$ is added modulo each of the p_i, which breaks the security proof. Our technique is to replace the term $2r$ in (1) by another subset sum of public key elements which, taken modulo each of the p_i's, generate a lattice with special properties. We then apply the Leftover Hash Lemma modulo this lattice instead of only modulo q_0, which proves semantic security.

In addition to componentwise addition and multiplication, we also show how to perform any permutation on plaintext bits publicly. As opposed to RLWE based schemes [BGV12,GHS12a], we cannot use an underlying algebraic structure to perform rotations over plaintext bits (clearly, the automorphisms of $\mathbb{Z}$ do not provide any useful action on ciphertexts). Instead we show how to perform arbitrary permutations on the plaintext vector during the ciphertext refresh operation at no additional cost (but with a slight increase of the public key size). Our ciphertext refresh operation $\mathsf{Recrypt}$ is done in parallel over the ℓ slots, with the same complexity as a single $\mathsf{Recrypt}$ operation in the original scheme.

Finally, we describe an implementation of our batch DGHV scheme of Section 4, with concrete parameters. We perform an homomorphic evaluation of the full AES encryption circuit. For the "Large" parameters with 72 bits of security, our implementation homomorphically encrypts up to 531 AES ciphertexts in parallel in an amortized time of 12 minutes per AES ciphertext on a desktop computer. This is comparable to the timings presented by Gentry *et al.* at Crypto 2012 for their implementation of an RLWE-based scheme [GHS12b].[8]

While our batch variant of DGHV does not provide additional features nor significantly improved efficiency over the RLWE-based scheme of [GHS12a], we believe it is interesting to obtain FHE schemes with similar properties but based on different techniques and assumptions.

2 The Somewhat Homomorphic DGHV Scheme

We first recall the somewhat homomorphic encryption scheme over the integers of van Dijk, Gentry, Halevi and Vaikuntanathan (DGHV) in [DGHV10]. Let λ be the security parameter, τ be the number of elements in the public key, γ their bit-length, η the bit-length of the secret key p and ρ (resp. ρ') the bit-length of the noise in the public key (resp. in a fresh ciphertext).

For a real number x, we denote by $\lceil x \rceil$, $\lfloor x \rfloor$, and $\lceil x \rfloor$ the upper, lower, and nearest integer part of x. For integers z, p we denote the reduction of z modulo p by $(z \bmod p)$ or $[z]_p$ with $-p/2 < [z]_p \leqslant p/2$.

[8] Notice that our implementation uses bootstrapping whereas the implementation of [GHS12b] used a leveled homomorphic encryption scheme without bootstrapping.

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

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

DGHV. $\mathsf{KeyGen}(1^\lambda)$**.** Generate an η-bit random prime integer p. For $0 \leqslant i \leqslant \tau$, 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 $\mathsf{pk} = (x_0, x_1, \ldots x_\tau)$ and $\mathsf{sk} = p$.

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

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

DGHV. $\mathsf{Evaluate}(\mathsf{pk}, C, c_1, \ldots, c_t)$**.** 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.

DGHV. $\mathsf{Decrypt}(\mathsf{sk}, c)$**.** Output $m \leftarrow [c \bmod p]_2$.

As shown in [DGHV10] the 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 $m + m' \bmod 2$ under a $(\rho' + 1)$-bit noise and the ciphertext $c \cdot c'$ is an encryption of $m \cdot m'$ with noise bit-length $\simeq 2\rho'$. Since the ciphertext noise must remain smaller than p to maintain correctness, the scheme roughly allows η/ρ' successive multiplications on ciphertexts. The scheme is semantically secure under the Approximate-GCD assumption (see [DGHV10]):

Definition 1 (Approximate GCD). *The (ρ, η, γ)-approximate GCD problem consists, given a random η-bit odd integer p and given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$, in outputting p.*

3 Batch DGHV Scheme Based on a new Decisional Assumption[9]

We describe our first extension of the DGHV scheme for the batch setting. We extend the DGHV scheme by packing ℓ plaintexts $m_0, \ldots, m_{\ell-1}$ into a single ciphertext, using the Chinese Remainder Theorem. Moreover, for somewhat homomorphic encryption, this allows us to encrypt not only bits but elements from rings of form $\mathbb{Z}_Q$.

[9] A part of this section was made public through [Mem12].

Therefore, for public parameters $Q_0, \ldots, Q_{\ell-1}$, we encrypt $(m_0, \ldots, m_{\ell-1}) \in \mathbb{Z}_{Q_0} \times \cdots \times \mathbb{Z}_{Q_{\ell-1}}$ into a ciphertext of the following form:

$$c = \mathsf{CRT}_{q_0,p_0,\ldots,p_{\ell-1}}(q, Q_0 r_0 + m_0, \ldots, Q_{\ell-1} r_{\ell-1} + m_{\ell-1}),$$

where q is uniform random modulo q_0 and r_i's are small noises. Decryption can be done by

$$m_i = [c \bmod p_i]_{Q_i}.$$

Homomorphic addition and multiplication is done by the corresponding arithmetic operations on ciphertexts.

This scheme can be considered as encrypting vectors $(m_0, \ldots, m_{\ell-1})$ and supporting parallel, componentwise additions and multiplications. Or, if we choose the Q_i's to be pairwise coprime, then using the isomorphism $\mathbb{Z}_{\prod Q_i} \cong \prod \mathbb{Z}_{Q_i}$, we may regard this as a somewhat homomorphic encryption supporting arithmetic operations on $\mathbb{Z}_Q$, with $Q = \prod Q_i$.

In order to allow public-key encryption, we provide integers x'_i and x_i in the public key, such that $x'_i \bmod p_j = Q_j r'_{i,j} + \delta_{i,j}$, and $x_i \bmod p_j = Q_j r_{i,j}$ for all i, j where $\delta_{i,j}$ is the Kronecker delta. Then, a plaintext vector $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ is encrypted as follows:

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i \in S} x_i \right]_{x_0}.$$

As explained in the introduction, the additional, larger noise $2r$ from the DGHV scheme is not used in this version. This simplifies the construction, with the trade-off of a new decisional assumption and a larger security loss. In Section 4, we present another version which handles this issue differently. In Section 5, we describe a transformation to fully homomorphic encryption schemes where $Q_0 = \cdots = Q_{\ell-1} = 2$.

3.1 Description

IDGHV. $\mathsf{KeyGen}(1^\lambda, (Q_j)_{0 \leqslant j < \ell})$**.** Choose η-bit distinct primes p_j, $0 \leqslant j < \ell$, and denote π their product. Let us define the error-free public key element $x_0 = q_0 \cdot \pi$, where $q_0 \leftarrow \mathbb{Z} \cap [0, 2^\gamma/\pi)$ is a 2^{λ^2}-rough integer.[10] Make sure that $\gcd(Q_j, x_0) = 1$ for $0 \leqslant j < \ell$, and abort otherwise.[11]

Choose the following integers x_i and x'_i with a quotient by π uniformly and independently distributed in $\mathbb{Z} \cap [0, q_0)$, and with the following distribution modulo p_j for $0 \leqslant j < \ell$:

$$\begin{aligned} &1 \leqslant i \leqslant \tau, && x_i \bmod p_j = Q_j r_{i,j}, && r_{i,j} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho), \\ &0 \leqslant i \leqslant \ell - 1, && x'_i \bmod p_j = Q_j r'_{i,j} + \delta_{i,j}, && r'_{i,j} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho). \end{aligned}$$

[10] An integer a is b-rough when it does not contain prime factors smaller than b. As in [CMNT11] one can generate q_0 as a product of 2^{λ^2}-bit primes.

[11] Note that the abort case happens with negligible probability.

Finally, let $\mathsf{pk} = \left\{x_0, (Q_i)_{0\leqslant i\leqslant \ell-1}, (x_i)_{1\leqslant i\leqslant \tau}, (x'_i)_{0\leqslant i\leqslant \ell-1}\right\}$ and let $\mathsf{sk} = (p_j)_{0\leqslant j\leqslant \ell-1}$.

IDGHV.$\mathsf{Encrypt}(\mathsf{pk}, \boldsymbol{m})$. For any $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ with $m_i \in \mathbb{Z}_{Q_i}$, choose a random binary vector $\boldsymbol{b} = (b_i)_{1\leqslant i\leqslant \tau} \in \{0,1\}^\tau$ and output the ciphertext:

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i=1}^{\tau} b_i \cdot x_i\right]_{x_0}. \tag{3}$$

IDGHV.$\mathsf{Decrypt}(\mathsf{sk}, c)$. Output $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ where $m_j \leftarrow [c \bmod p_j]_{Q_j}$.

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

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

3.2 Parameters and Correctness

The size of the message space is determined by ℓ and the binary length of the Q_j's, which can be an integer from 2 to $\eta/8$ depending on the multiplicative depth of the scheme. The parameters should satisfy the following constraints:

- $\rho = \tilde{\Omega}(\lambda)$, to be secure against Chen-Nguyen's attack [CN12] and Howgrave-Graham's attack [HG01],
- $\eta = \tilde{\Omega}(\lambda^2 + \rho \cdot \lambda)$, to resist the factoring attack using the elliptic curve method [Len87], and to permit enough multiplicative depth,
- $\gamma = \omega(\eta^2 \log \lambda)$, to resist Cohn and Heninger's attack [CH11] and the attack using Lagarias algorithm [Lag85] on the approximate GCD problem,
- $\tau = \gamma + \omega(\log \lambda)$, in order to use leftover hash lemma (see Section 3.3).

We choose $\gamma = \tilde{\mathcal{O}}(\lambda^5)$, $\eta = \tilde{\mathcal{O}}(\lambda^2)$, $\rho = 2\lambda$, $\tau = \gamma + \lambda$ which is similar to the DGHV's convenient parameter setting [DGHV10]. We remark that b_i can be chosen in a much larger interval so as to reduce the public key size as in Section 4.

Notice that the above scheme is correct as is proved in the full version [KLYC13] with the definition of correctness for homomorphic encryption schemes, with respect to a set of permitted circuits.

3.3 Semantic Security

Here we show the semantic security of the IDGHV scheme, based on a new assumption called ℓ-$\mathbf{DACD}_Q$. In fact, this rather complicated assumption is given only as an intermediate step for the proof, and in Section 3.4, we prove this assumption from a simpler decisional assumption, called the Decisional Approximate GCD assumption (**DACD**). So the security of our scheme is eventually based on this assumption.

Given two integer vectors $\boldsymbol{p} = (p_0, \ldots, p_{\ell-1})$, $\boldsymbol{Q} = (Q_0, \ldots, Q_{\ell-1})$ of length ℓ and an integer q_0, let us define the distribution $\mathcal{D}_\rho(\boldsymbol{p}; \boldsymbol{Q}; q_0)$ as follows. The output of the distribution is

$$x = \mathsf{CRT}_{q_0, p_0, \ldots, p_{\ell-1}}(q, Q_0 r_0, \ldots, Q_{\ell-1} r_{\ell-1}),$$

where $q \leftarrow \mathbb{Z} \cap [0, q_0)$ and $r_i \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ for $i = 0, \ldots, \ell - 1$.

Definition 2 (ℓ-Decisional Approximate GCD_Q Problem: ℓ-DACD_Q). *The $(\rho, \eta, \gamma, \mu)$-$\ell$-decisional approximate GCD_Q problem is: for η-bit distinct primes $p_0, \ldots, p_{\ell-1}$ and μ-bit integers $Q_0, \ldots, Q_{\ell-1}$, given a γ-bit integer $x_0 := q_0 p_0 \cdots p_{\ell-1}$, with $\gcd(x_0, Q_i) = 1$ for $i = 0, \ldots, \ell - 1$, and polynomially many samples from $\mathcal{D} := \mathcal{D}_\rho(\boldsymbol{p}; \boldsymbol{Q}; q_0)$ and a set X consisting of ℓ integers $x'_i = \mathsf{CRT}_{q_0, p_0, \ldots, p_{\ell-1}}(q_i, Q_0 r'_{i,0} + \delta_{i,0}, \ldots, Q_{\ell-1} r'_{i,\ell-1} + \delta_{i,\ell-1})$ where q_i, $r'_{i,j}$ are chosen as $q_i \leftarrow \mathbb{Z} \cap [0, q_0)$, $r'_{i,j} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ for all $i, j \in \{0, \ldots, \ell - 1\}$, determine $b \in \{0, 1\}$ from $z = x + r \cdot b \mod x_0$ where $x \leftarrow \mathcal{D}$ and $r \leftarrow \mathbb{Z} \cap [0, x_0)$.*

The $(\rho, \eta, \gamma, \mu)$-$\ell$-decisional approximate GCD_Q assumption then states that this problem is hard for any polynomial time distinguisher.

Theorem 1. *The* IDGHV *scheme is semantically secure under the ℓ-decisional approximate GCD_Q assumption.*

Proof. We provide a sketch of the proof. See the complete proof in the full version [KLYC13].

Essentially, the idea of the proof is that the IDGHV scheme is a lossy encryption [BHY09]. Both the correctly generated public key pk and the lossy key pk' have the following form

$$\left\{ x_0, (Q_i)_{0 \leqslant i \leqslant \ell-1}, (x_i)_{1 \leqslant i \leqslant \tau}, (x'_i)_{0 \leqslant i \leqslant \ell-1} \right\},$$

but note that, for the real public key pk, x_i are chosen as $x_i \leftarrow \mathcal{D}_\rho(\boldsymbol{p}; \boldsymbol{Q}; q_0)$, and for the lossy public key pk', $x_i \leftarrow \mathbb{Z} \cap [0, x_0)$.

Now we may rely on the standard hybrid argument to show that pk and pk' are computationally indistinguishable, under the $(\rho, \eta, \gamma, \mu)$-$\ell$-decisional approximate GCD_Q assumption: for each $\hat{\imath} \in \{1, \ldots, \tau\}$, we define $\mathsf{pk}^{(\hat{\imath})}$, where

$$x_i \leftarrow \begin{cases} \mathbb{Z} \cap [0, x_0) & \text{for } 1 \leqslant i \leqslant \hat{\imath}, \\ \mathcal{D}_\rho(\boldsymbol{p}; \boldsymbol{Q}; q_0) & \text{for } \hat{\imath} < i \leqslant \tau. \end{cases}$$

Then $\mathsf{pk}^{(0)}$ is identical to pk, and $\mathsf{pk}^{(\tau)}$ is identical to pk', and using the $(\rho, \eta, \gamma, \mu)$-$\ell$-decisional approximate GCD_Q assumption, we may show that $\mathsf{pk}^{(\hat{\imath})}$ is indistinguishable from $\mathsf{pk}^{(\hat{\imath}+1)}$ for $\hat{\imath} = 0, \ldots, \tau - 1$, proving that pk and pk' are computationally indistinguishable.

Next, we show that under the lossy key pk', the IDGHV scheme is semantically secure: for any two plaintexts $\boldsymbol{m}$ and $\boldsymbol{m}'$, the distributions of their corresponding ciphertexts c, c' under the lossy key pk' are statistically close. Namely, recall that from Equation (3), we have

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i=1}^{\tau} b_i \cdot x_i \right]_{x_0}.$$

Now, when we choose x_i from the lossy public key pk', x_i are uniform on $\mathbb{Z} \cap [0, x_0)$, and we may use the Leftover Hash Lemma. In fact we may use the simplified version shown in Lemma 1 of [DGHV10] to conclude that the distribution of $[\sum_{i=1}^{\tau} b_i \cdot x_i]_{x_0}$ is statistically close to the uniform distribution on $\mathbb{Z} \cap [0, x_0)$, when the parameters are chosen according to Section 3.2. Therefore, the distribution of c is uniformly random, regardless of the plaintext vector $\boldsymbol{m}$. This shows that IDGHV is a lossy encryption scheme, and the semantic security directly follows from that. □

3.4 Hardness Assumption

We show that the semantic security of our scheme can be based on a simpler decisional assumption with a single prime p. For two specific integers p and q_0, we use the following distribution over γ-bit integers:

$$\mathcal{D}_\rho(p, q_0) := \{\mathsf{Choose}\ q \leftarrow [0, q_0), r \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho) : \mathsf{Output}\ y = q \cdot p + r\}.$$

Definition 3 (Decisional Approximate GCD Problem: DACD). *The (ρ, η, γ)-decisional approximate GCD problem is: for a random η-bit prime p, given a γ-bit integer $x_0 = q_0 \cdot p$ and polynomially many samples from $\mathcal{D}_\rho(p, q_0)$, determine $b \in \{0, 1\}$ from $z = x + r \cdot b \mod x_0$ where $x \leftarrow \mathcal{D}_\rho(p, q_0)$ and $r \leftarrow \mathbb{Z} \cap [0, x_0)$.*

The (ρ, η, γ)-decisional approximate GCD assumption then states that this problem is hard for any polynomial time distinguisher.

We now give a sketch of the proof of the following lemma. For the detailed proof, see the full version [KLYC13].

Lemma 1. *The ℓ-decisional approximate GCD_Q problem is hard under the decisional approximate GCD assumption.*

Proof (Sketch). The main difference between **DACD** and 1-**DACD**$_Q$ is the existence of Q_0 in the latter problem. When Q_0 is coprime to x_0, it is easy to see that both problems are equivalent. Namely, multiplying by Q_0 modulo x_0 efficiently converts samples from $\mathcal{D}_\rho(p, q_0)$ to $\mathcal{D}_\rho((p); (Q_0); q_0)$.

Now, we need to show that the ℓ-**DACD**$_Q$ problem is hard under the 1-**DACD**$_Q$ assumption. We use a hybrid argument. From the 1-**DACD**$_Q$ problem instance $x_0 = q_0 \cdot p$ and samples from $\mathcal{D}_\rho((p); (Q_0); q_0)$, we choose $\ell - 1$ primes ourselves. Putting a prime p from 1-**DACD**$_Q$ at a random position i_0 among the p_i's, we can construct samples from $\mathcal{D}_\rho(\boldsymbol{p}; \boldsymbol{Q}; q_0)$ using the Chinese Remainder Theorem with samples from $\mathcal{D}_\rho((p); (Q_0); q_0)$. And a set X also can be efficiently constructed. For the challenge z, we construct a ℓ-**DACD**$_Q$ challenge z' such that

$$z' = \mathsf{CRT}_{x_0, (p_i)_{i \neq i_0}}(z, r'_0 Q_0, \ldots, r'_{i_0 - 1} Q_{i_0 - 1}, e'_{i_0 + 1}, \ldots, e'_{\ell - 1})$$

where $r'_i \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ and $e'_i \leftarrow \mathbb{Z} \cap [0, p_i)$. By the hybrid argument, it can be shown that any ℓ-**DACD**$_Q$ distinguisher can be efficiently converted to a 1-**DACD**$_Q$ distinguisher. This terminates the proof of Lemma 1. □

Corollary 1. *The* IDGHV *scheme is semantically secure under the decisional approximate GCD assumption.*

3.5 Application to Secure Large Integer Arithmetic

Secure integer arithmetic is one of the most important applications of homomorphic encryption schemes. It includes frequently used statistical functions such as mean, standard deviation, logistical regression, and secure evaluation of a multivariate function over the integers. Some applications may require very large integer inputs in the computation of these functions. For the homomorphic computation of these functions, one may use FHE supporting homomorphic bit operations. However, the large ciphertext expansion and rather high cost of bootstrapping make this cumbersome and inefficient. In fact, even an addition of two λ-bit integers using bit operations needs computing degree-$O(\lambda)$ polynomial over $\mathbb{Z}_2$ due to the carry computation. For this reason, it is very important to construct an efficient somewhat homomorphic scheme supporting large integer arithmetic on encrypted data.

As mentioned earlier in this section, IDGHV scheme supports arithmetic operations on $\mathbb{Z}_Q$ with $Q = \prod_{i=0}^{\ell-1} Q_i$ when all Q_i's are pairwise coprime. We can freely choose ℓ up to $\tilde{\mathcal{O}}(\lambda^3)$ depending on the applications. And the advantage of our scheme in the overhead stands out, as the plaintext space gets larger.

4 Batch DGHV Scheme Based on the Error-Free Approximate-GCD

In this section, we present a variant of the previous IDGHV scheme but based on a (weaker) *computational* assumption instead of the decisional assumption from Def. 3. This is made possible by adding a new set of elements in the public key but yields a more intricate scheme.[12]

Ideally we would like to base the security of the new batch DGHV scheme on the same assumption as the original single-bit DGHV scheme, *i.e.* the Approximate-GCD assumption from Definition 1. However we can only show its security under the (stronger) Error-Free Approximate-GCD assumption already considered in [DGHV10,CMNT11,CNT12]. For two specific integers p and q_0, we use the following distribution over γ-bit integers:

$$\mathcal{D}_\rho(p, q_0) = \{\mathsf{Choose}\ q \leftarrow [0, q_0), r \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho) : \mathsf{Output}\ y = q \cdot p + r\}\,.$$

Definition 4 (Error-free approximate GCD). *The (ρ, η, γ)-error-free approximate-GCD problem is: For a random η-bit prime p, given $y_0 = q_0 \cdot p$ where q_0 is a random integer in $[0, 2^\gamma/p)$, and polynomially many samples from $\mathcal{D}_\rho(p, q_0)$, output p.*

[12] For the sake of simplicity, we use $Q_0 = \cdots = Q_{\ell-1} = 2$ throughout the rest of the paper; however the security proof extends to general Q_i's as in Section 3.

In the following we briefly explain our proof strategy. In the original DGHV scheme, public-key encryption is performed by masking the message m with a random subset sum of the public key elements $x_j = q_j \cdot p + r_j$ as

$$c = \left[m + 2r + 2 \sum_{j \in S} x_j \right]_{x_0} . \tag{4}$$

The semantic security is proved by applying the Leftover Hash Lemma on the subset sum modulo q_0, and using the random $2r$ in (4) to further randomize the ciphertext modulo p.

However in the batch scheme it would not be sufficient to add such random term $2r$; namely the *same* random $2r = 2r \bmod p_i$ would be added modulo each of the p_i's, whereas for the security proof to go through these random terms should be independently distributed modulo each of the p_i's. Therefore a new technique is required to extend DGHV to a semantically secure batch encryption scheme, whose security can be based on the Error-Free Approximate-GCD problem.

In the following, we start by describing a variant of DGHV still for a single bit message m only, but which *does* extend naturally to the batch setting. We first consider the DGHV scheme without the additional random $2r$, since this term is of no use in the batch setting. A single message bit m is then encrypted as

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

where $x_i = q_i \cdot p + r_i$. In order to prove semantic security as in [DGHV10], one should prove that the values q and r' given by $c = q \cdot p + 2r' + m$ are essentially random and independently distributed. The randomness of $q = 2 \sum_{i \in S} q_i \bmod q_0$ follows from the Leftover Hash Lemma (LHL) modulo q_0. However we cannot apply the LHL to $r' = \sum_{i \in S} r_i$ because it is distributed over $\mathbb{Z}$ instead of modulo an integer. Note that in the original scheme the randomness of r' followed from adding a random $2r$ in (4), much larger than the r_i's.

Let us assume that we could somehow reduce the integer variable $r' = \sum_{i \in S} r_i$ modulo some integer ϖ. Then we could apply the LHL simultaneously modulo q_0 and modulo ϖ, and the distributions of $q \bmod q_0$ and $r' \bmod \varpi$ would be independently random as required. However, during public-key encryption we certainly do not have access to the variable $r' = \sum_{i \in S} r_i$, so we cannot *a priori* reduce it modulo an integer ϖ in the encryption phase.

Our technique is the following: instead of reducing the variable r' modulo ϖ, we add a large random multiple of ϖ to r'. This can be done by extending the public key with a new element Π such that $\Pi \bmod p = \varpi$. Encryption would then be performed as

$$c = \left[m + 2b \cdot \Pi + 2 \sum_{i \in S} x_i \right]_{x_0} \tag{5}$$

for some large random integer b. Modulo p this gives a new integer $r'' = r' + b \cdot \varpi$, and we argue that this enables to proceed as if r' was actually reduced modulo ϖ.

Namely, if we generate the r_i's such that the sum $r' = \sum_{i \in S} r_i$ is not much larger than ϖ, then reducing r' modulo ϖ would just subtract a small multiple of ϖ, which is negligible compared to the large random multiple $b \cdot \varpi$ obtained through (5). Formally the distribution of $r' + b \cdot \varpi$ is statistically close to $(r' \bmod \varpi) + b \cdot \varpi$, which enables us to apply the LHL to $r' \bmod \varpi$ and eventually obtain a security proof.

Now the advantage of (5) is that it can be easily extended to the batch setting. Instead of using a single random multiple of Π, we use a subset sum of ℓ such multiples Π_i, where $\Pi_i \bmod p_j = \varpi_{i,j}$. The Leftover Hash Lemma is then applied modulo the lattice generated by the $\varpi_{i,j}$. This shows that the random noise values modulo the p_i's follow essentially independent distributions, and eventually leads to a security proof based on the Error-Free Approximate-GCD problem above.

4.1 Description

BDGHV. $\mathsf{KeyGen}(1^\lambda)$. Generate a collection of ℓ random η-bit primes p_j, $0 \leqslant j < \ell$, and denote π their product. Let us define the error-free public key element $x_0 = q_0 \cdot \pi$, where $q_0 \leftarrow \mathbb{Z} \cap [0, 2^\gamma/\pi)$ is a 2^{λ^2}-rough integer.
Generate the following integers x_i, x'_i and Π_i with a quotient by π uniformly and independently distributed in $\mathbb{Z} \cap [0, q_0)$, and with the following distribution modulo p_j for $0 \leqslant j < \ell$:

$$\begin{aligned} &1 \leqslant i \leqslant \tau, && x_i \bmod p_j = 2r_{i,j}, \\ &0 \leqslant i \leqslant \ell - 1, && x'_i \bmod p_j = 2r'_{i,j} + \delta_{i,j}, \\ &0 \leqslant i \leqslant \ell - 1, && \Pi_i \bmod p_j = 2\varpi_{i,j} + \delta_{i,j} \cdot 2^{\rho'+1}, \end{aligned}$$

with $r_{i,j} \leftarrow \mathbb{Z} \cap (-2^{\rho'-1}, 2^{\rho'-1})$ and $r'_{i,j}, \varpi_{i,j} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$. Finally, let

$$\mathsf{pk} = \left\{ x_0, (x_i)_{1 \leqslant i \leqslant \tau}, (x'_i)_{0 \leqslant i \leqslant \ell-1}, (\Pi_i)_{0 \leqslant i \leqslant \ell-1} \right\}$$

and $\mathsf{sk} = (p_j)_{0 \leqslant j \leqslant \ell-1}$.

BDGHV. $\mathsf{Encrypt}(\mathsf{pk}, \boldsymbol{m} \in \{0,1\}^\ell)$. Choose random integer vectors $\boldsymbol{b} = (b_i) \in (-2^\alpha, 2^\alpha)^\tau$ and $\boldsymbol{b'} = (b'_i)_{0 \leqslant i \leqslant \ell-1} \in (-2^{\alpha'}, 2^{\alpha'})^\ell$ and output the ciphertext:

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i=0}^{\ell-1} b'_i \cdot \Pi_i + \sum_{i=1}^{\tau} b_i \cdot x_i \right]_{x_0}. \tag{6}$$

BDGHV. $\mathsf{Decrypt}(\mathsf{sk}, c)$. Output $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ where $m_j \leftarrow [c]_{p_j} \bmod 2$.

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

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

4.2 Parameters and Correctness

The parameters must meet the following constraints (where λ is the security parameter):

- $\rho = \Omega(\lambda)$ to avoid brute force attack on the noise [CN12,CNT12],
- $\eta \geqslant \alpha' + \rho' + 1 + \log_2(\ell)$ for correct decryption,
- $\eta \geqslant \rho \cdot \Theta(\lambda \log^2 \lambda)$ for homomorphically evaluating the "squashed decryption" circuit
- $\gamma = \omega(\eta^2 \cdot \log \lambda)$ in order to thwart lattice-based attacks [DGHV10,CMNT11];
- $\rho' \geqslant \rho + \lambda$ and $\alpha' \geqslant \alpha + \lambda$ for the proof of semantic security,
- $\alpha \cdot \tau \geqslant \gamma + \lambda$ and $\tau \geqslant \ell \cdot (\rho' + 2) + \lambda$ in order to apply the leftover hash lemma.

To satisfy the above constraints one can take $\rho = 2\lambda$, $\eta = \tilde{\mathcal{O}}(\lambda^2)$, $\gamma = \tilde{\mathcal{O}}(\lambda^5)$, $\alpha = \tilde{\mathcal{O}}(\lambda^2)$, $\tau = \tilde{\mathcal{O}}(\lambda^3)$ as in [CNT12], with $\rho' = \tilde{\mathcal{O}}(\lambda)$, $\alpha' = \tilde{\mathcal{O}}(\lambda^2)$ and $\ell = \tilde{\mathcal{O}}(\lambda^2)$. We refer to Section 5.4 for concrete parameters and timings. We show in [CLT13, Appendix A] that the above scheme is correct for a set of permitted circuits.

4.3 Semantic Security

To prove the semantic security of our scheme, we first introduce a temporary decisional assumption that is implied by the Error-Free Approximate-GCD assumption.

Given integers q_0 and $p_0, \ldots, p_{\ell-1}$, we define the oracle $\mathcal{O}_{q_0,(p_i)}(\boldsymbol{v})$ which, given as input a vector $\boldsymbol{v} \in \mathbb{Z}^\ell$, outputs x with

$$x = \mathsf{CRT}_{q_0,(p_i)}(q, v_0 + 2r_0, \ldots, v_{\ell-1} + 2r_{\ell-1})$$

where $q \leftarrow [0, q_0)$ and $r_i \leftarrow (-2^\rho, 2^\rho)$. Therefore $\mathcal{O}_{q_0,(p_i)}(\boldsymbol{v})$ outputs a ciphertext for the plaintext $\boldsymbol{v}$. Note that the components v_i can be any integer, not only $0, 1$.

Definition 5 (O-ℓ-dAGCD$_{\lambda,\gamma,\eta}$). *The oracle ℓ-decisional-approximate-GCD problem is as follows. Pick random η-bit integers $p_0, \ldots, p_{\ell-1}$ of product π, a random 2^{λ^2}-rough $q_0 \leftarrow \mathbb{Z} \cap [0, 2^\gamma/\pi)$, a random bit b, set $\boldsymbol{v}_0 = (0, \ldots, 0)$ and $\boldsymbol{v}_1 \leftarrow \{0,1\}^\ell$. Given $x_0 = q_0 p_0 \cdots p_{\ell-1}$, $z = \mathcal{O}_{q_0,(p_i)}(\boldsymbol{v}_b)$ and oracle access to $\mathcal{O}_{q_0,(p_i)}$, guess b.*

This decisional problem is somehow to distinguish between an encryption of 0 and an encryption of a random message. To prove the semantic security of our scheme, we must show that this still holds when using the public-key encryption procedure instead of the oracle $\mathcal{O}_{q_0,(p_i)}$; this essentially amounts to applying a variant of the Leftover Hash Lemma. We refer to the full version [CLT13] for the proof.

Theorem 2. *The batch DGHV scheme is semantically secure under the oracle ℓ-decisional-approximate-GCD assumption.*

Lemma 2. *The oracle-ℓ-decisional-approximate-GCD problem is hard if the error-free-approximate-GCD problem is hard.*

Corollary 2. *The batch DGHV scheme is semantically secure under the error-free-approximate-GCD assumption.*

5 Making the Scheme Fully Homomorphic

In this section, we follow Gentry's blueprint [Gen09] to transform a somewhat homomorphic encryption scheme into a fully homomorphic encryption scheme. This technique applies directly to both schemes described in Section 3 and 4.

5.1 The Squashed Scheme

As mentioned in the introduction, to follow Gentry's blueprint and make our somewhat homomorphic schemes amenable to bootstrapping, we first need to squash the decryption circuit, *i.e.* change the decryption procedure so as to express it as a low degree polynomial in the bits of the secret key.

We use the same technique as in the original DGHV scheme [DGHV10] but generalize it to the batch setting. We add to the public key a set $\boldsymbol{y} = \{y_0, \ldots, y_{\Theta-1}\}$ of rational numbers in $[0, 2)$ with κ bits of precision after the binary point, such that for all $0 \leqslant j \leqslant \ell - 1$ there exists a sparse subset $S_j \subset [0, \Theta - 1]$ of size θ with $\sum_{i \in S_j} y_i \simeq 1/p_j \bmod 2$. The secret-key is replaced by the indicator vector of the subsets S_j. Formally the scheme is modified as follows:

$\mathsf{BDGHV.KeyGen}(1^\lambda)$**.** Generate $\mathsf{sk}^* = (p_0, \ldots, p_{\ell-1})$ and pk^* as before. Set $x_{p_j} \leftarrow \lfloor 2^\kappa / p_j \rceil$ for $j = 0, \ldots, \ell - 1$. Choose at random Θ-bit vectors $\boldsymbol{s}_j = (s_{j,0}, \ldots, s_{j,\Theta-1})$, each of Hamming weight θ, for $0 \leqslant j < \ell$. Choose at random Θ integers $u_i \in [0, 2^{\kappa+1})$ for $0 \leqslant i < \Theta$, fulfilling the condition that $x_{p_j} = \sum_{i=0}^{\Theta-1} s_{j,i} \cdot u_i \bmod 2^{\kappa+1}$ for all j. Set $y_i = u_i / 2^\kappa$ and $\boldsymbol{y} = (y_0, \ldots, y_{\Theta-1})$. Hence, each y_i is a positive number smaller than two, with κ bits of precision after the binary point, and verifies

$$\frac{1}{p_j} = \sum_{i=0}^{\Theta-1} s_{j,i} \cdot y_i + \varepsilon_j \bmod 2 \tag{7}$$

for some $|\varepsilon_j| < 2^{-\kappa}$. Finally, output the key pair

$$\mathsf{sk} = (\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{\ell-1}) \quad \text{and} \quad \mathsf{pk} = (\mathsf{pk}^*, y_0, \ldots, y_{\Theta-1}) \,.$$

BDGHV. $\mathsf{Expand}(\mathsf{pk}, c)$**.** The ciphertext expansion procedure takes as input a ciphertext c and computes an expanded ciphertext: for every $0 \leqslant i \leqslant \Theta - 1$, compute z_i given by $z_i = \lfloor c \cdot y_i \rceil \bmod 2$ with n bits of precision after the binary point. Define the vector $\boldsymbol{z} = (z_i)_{i=0,\ldots,\Theta-1}$ and output the expanded ciphertext $(c, \boldsymbol{z})$.

BDGHV. $\mathsf{Decrypt}(\mathsf{sk}, c, \boldsymbol{z})$**.** Output $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ with

$$m_j \leftarrow \left[\left\lfloor \sum_{i=0}^{\Theta-1} s_{j,i} \cdot z_i \right\rceil \right]_2 \oplus (c \bmod 2) \; . \tag{8}$$

This completes the description of the scheme. We use $n = \lceil \log_2(\theta + 1) \rceil$ as in [CMNT11].

5.2 Bootstrapping

As in [DGHV10], we get that the BDGHV scheme is bootstrappable. Moreover, the Recrypt procedures works naturally in *parallel* over the plaintext bits.

Namely the decryption equation (8) for the batch scheme can be evaluated homomorphically by providing for all $0 \leqslant i < \Theta$ an encryption σ_i of the ℓ secret-key bits $s_{j,i}$, with:

$$\sigma_i = \mathsf{Encrypt}(s_{0,i}, \ldots, s_{\ell-1,i}) \; .$$

This gives a new ciphertext that encrypts the same ℓ-bit plaintext vector, but with a (possibly) reduced noise. Notice that the ℓ equations in (8) are homomorphically evaluated in parallel, one in each of the ℓ plaintext slots of the ciphertext. Therefore, with the same complexity as a single Recrypt operation in the original scheme, the batch Recrypt operation is performed in parallel over the ℓ slots.

From Gentry's theorem, we obtain a homomorphic encryption scheme for circuits of any depth. The proof of the following theorem is identical to the proof of [CMNT11, Theorem 5.1].

Theorem 3. *Let $\mathcal{E}$ be the above scheme, and let $D_{\mathcal{E}}$ be the set of augmented (squashed) decryption circuits. Then $D_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$.*

5.3 Complete Set of Operations for Plaintext Vectors

From what precedes, we can implement homomorphic SIMD-type operations on our packed ciphertexts, where the Add and Mult operations are applied to ℓ different input bits at once. However, a desired feature when dealing with packed ciphertexts is the ability to move values between plaintext slots with a public Permute operation. As opposed to [GHS12a] we cannot rely on an underlying algebraic structure. Instead we show how to perform such Permute at ciphertext refresh time, *i.e.* when performing a Recrypt. This feature is therefore supported at no extra cost assuming a ciphertext refresh operation has to be carried out anyway (*i.e.* after each Mult gate). Notice that a similar technique

was independently described in [BGH13] for the RLWE-based fully homomorphic schemes [BV11a,BV11b,GHS12a].

For any permutation ζ over $\{0,\ldots,\ell-1\}$, we want to homomorphically evaluate the function

$$\ell\text{-}\mathsf{Permute}\,(\zeta,(u_0,\ldots,u_{\ell-1})) = \left(u_{\zeta(0)},\ldots,u_{\zeta(\ell-1)}\right).$$

Let ζ be a permutation to be applied homomorphically on the plaintext bits. During the KeyGen operation, the authority can define for each $i \in [0,\Theta-1]$

$$\sigma_i^\zeta = \mathsf{Encrypt}(s_{\zeta(0),i},\ldots,s_{\zeta(\ell-1),i}).$$

Now, performing the ciphertext refresh operation ("recryption") with the σ_i^ζ's instead of the σ_i's gives a ciphertext of the plaintext vector $(m_{\zeta(0)},\ldots,m_{\zeta(\ell-1)})$ which is exactly the desired result. Therefore any permutation ζ can be implemented by putting the corresponding σ_i^ζ's in the public key.

To be able to perform arbitrary permutations on the plaintext vector, one can augment the public key by a minimal set of permutations ζ's that generates the whole permutation group $\mathfrak{S}_\ell$ over $\{0,\ldots,\ell-1\}$, such as the transposition $(1,2)$ and the cycle $(1,2,\ldots,\ell)$. In that case the impact on the public key is small (as only $2\cdot\Theta\cdot\gamma$ bits are added), but the performance overhead is significant, since as many as $\mathcal{O}(\ell)$ ciphertext refresh operations may be needed to carry out a desired permutation.

A more practical solution is to use a Beneš network [Ben64] of permutations as in [GHS12a]. In that case it suffices to add $2\log_2(\ell)$ permuting elements to the public key to enable circular rotations by $\pm 2^i$ bit position. Then any permutation can be obtained in $(2\log(\ell)-1)$ steps. At each step, at most two rotations and two Select operations are performed, where the Select operation on c_1 and c_2 constructs a ciphertext where each of the ℓ plaintext slot is chosen either from c_1 or c_2; such Select operation is easily obtained with two Mult (and two recryptions) and one Add, see [GHS12a]. This approach has a limited impact on the public key ($2\log_2(\ell)\cdot\Theta\cdot\gamma$ more bits), and any permutation can then be performed with at most $6\cdot(2\log_2\ell-1)$ recryptions.

In practice, however, the circuit to be homomorphically evaluated is likely to be known in advance, so it is possible to put a set of distinguished permutations in the public key that provides an optimal time-memory trade-off. In the next section, we describe two variants of homomorphic evaluations of the full AES circuit that require respectively only *four* permutations and no permutation at all.

5.4 Implementation Results

We provide in Table 1 concrete key sizes and timings for the batch DGHV scheme, based on a C++ implementation using the GMP library. We use essentially the same parameters as in [CNT12,CT12]; in particular, the parameters take into account the attack from [CN12]. We use the same compressed public-key variant

as in [CNT12]; a complete description of the scheme is given in [CLT13]. As in [CMNT11,CNT12], we take $n = 4$ and $\theta = 15$ for all security levels.

We obtain essentially the same running times as in [CNT12]. The main difference is that the Recrypt operation is now performed in parallel over $\ell = 531$ bits (for the "Large" setting) instead of a single bit.

Instance	λ	ℓ	ρ	η	$\gamma/10^6$	τ	Θ	pk size	KeyGen	Encrypt	Decrypt	Expand	Recrypt
Small	52	37	41	1558	0.90	661	555	13 MB	1.74s	0.23s	0.02s	0.08s	1.10s
Medium	62	138	56	2128	4.6	2410	2070	304 MB	73s	3.67s	0.45s	1.60s	11.9s
Large	72	531	71	2698	21	8713	7965	5.6 GB	3493s	61s	9.8s	28s	172s

Table 1. Benchmarking for our Batch DGHV with a compressed public key on a desktop computer (Intel Core i7 at 3.4Ghz, 32GB RAM).

6 Homomorphic Evaluation of the AES Circuit

In this section, we show how to homomorphically evaluate the AES-128 encryption circuit using the batch encryption scheme of Section 4 with compressed public key elements (see [CLT13]), and provide concrete timings. A similar implementation with the RLWE-based fully-homomorphic encryption scheme [BV11a,BV11b,GHS12a] was already described in [GHS12b]. As mentioned in [SV11,NLV11,GHS12b], such an implementation can be used to optimize the communication cost in cloud-based applications. Indeed, since the ciphertext expansion ratio in most fully-homomorphic encryption schemes is huge, data can rather be sent encrypted under AES with a ciphertext expansion equal to 1, along with the public key $\mathsf{pk}_{\mathsf{FHE}}$ of the FHE scheme as well as the AES secret-key encrypted under $\mathsf{pk}_{\mathsf{FHE}}$. Then, before the cloud performs homomorphic operations on the data, it can first run the AES decryption algorithm homomorphically to obtain the plaintext data encrypted under $\mathsf{pk}_{\mathsf{FHE}}$.

We consider our BDGHV scheme with ℓ slots. We describe two variants of our implementation which we call *byte-wise bitslicing* and *state-wise bitslicing*.

Byte-Wise Bitslicing. In this representation, the 16-byte AES state is viewed as a matrix of 16 rows of 8 bits each (one row for every byte). Each of the 8 columns is then stored on a different ciphertext. Therefore an AES state is stored in 8 ciphertexts, and one can perform $k = \ell/16$ AES encryptions in parallel using these 8 ciphertexts. Formally the AES state is composed of the ciphertexts $c_0, \ldots, c_7$, where the underlying plaintexts $\boldsymbol{m}_0, \ldots, \boldsymbol{m}_7$ are such that $\boldsymbol{m}_i[k \cdot 16 + j]$ is the i-th bit of the j-th element of the AES state of the k-th AES (see Figure 1).[13] We briefly describe how to implement the AES stages; full details on the implementation are given in [CLT13].

[13] Thus, $\boldsymbol{m}_0$ represents the LSBs of the AES states of the k AES-plaintexts, and $\boldsymbol{m}_7$ the MSBs. This construction is similar to general-purpose bitslicing [Bih97,KS09].

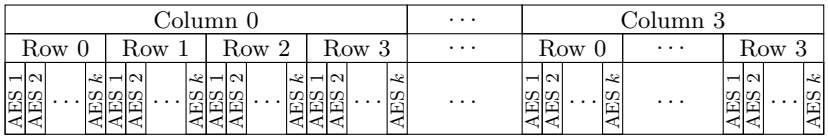

Fig. 1. Bit ordering in $\boldsymbol{m}_i$ in the byte-wise bitslicing representation

The AddRoundKey stage performs a XOR between the AES state and the current round key. This operation only consists of 8 Add operations. To minimize the number of Recrypt during the SubBytes stage, we used the 115 gates circuit of Boyar and Peralta [BP10] to compute the Sbox.[14] Thus, this step needs 17 Recrypt operations on 9 of the temporary variables and on the 8 outputs. In total, this stage costs 83 Add, 32 Mult and 17 Recrypt.

The ShiftRows stage consists in performing a permutation of the state. For this we add the σ_i^ζ's of the associated permutation ζ in the public key, and the rotation is performed at no additional cost during the final Recrypt of the SubBytes stages. Finally the MixColumns stage requires 3 permutations of the AES state; this yields a total of $3 \times 8 = 24$ Recrypt and 38 Add, and the addition of the σ_i^ζ's of three permutations ζ to the public key.

In total, our byte-wise implementation of AES requires 1260 Add, 320 Mult, and 377 Recrypt.

State-Wise Bitslicing. In this representation, each of the 128 bits of the AES state is stored in a different ciphertext. One can then perform $k = \ell$ AES encryptions in parallel. This corresponds to a full bitslice implementation of AES. More precisely the AES state is composed of 128 ciphertexts $c_0, \ldots, c_{127}$, where the underlying plaintexts $\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{127}$ are such that $\boldsymbol{m}_{i+j\cdot 8}[k]$ is the i-th bit of the j-th byte of the state of the k-th AES.

The AddRoundKey stage requires 128 Add operations. The SubBytes stage is implemented using the same circuit as above. Since the circuit needs to be evaluated on each of the 16 bytes of the AES state, the stage costs $16 \times 83 = 1328$ Add, $16 \times 32 = 512$ Mult, and $16 \times 17 = 272$ Recrypt. The ShiftRows stage consists in performing a permutation of the state, and this is done by permuting the indices of bits in the homomorphic AES state at no additional cost. The MixColumns stage requires 608 Add. The total cost the AES evaluation is then 14688 Add, 5120 Mult and 2448 Recrypt.

Implementation Results. We implemented both variants using the concrete parameters from Table 1; our results are summarized in Table 2. The relative time is the total time of AES evaluation divided by the number of encryptions processed in parallel. Notice that the state-wise bitslicing variant yields better relative times.

Our timings are comparable to [GHS12b] for the RLWE-based scheme, where a relative time of 5 minutes per block is reported; the authors used a 24-core

[14] To minimize the number of bootstrappings in a given circuit we refer to [LP13].

(a) Timings for byte-wise representation

Instance	λ	ℓ	# of enc. in parallel	Add-RoundKey	ShiftRows & SubBytes	Mix-Columns	**Total AES** (in hours)	**Relative time**
Small	52	48	3	0.04s	21s	29s	**0.125**	2min 30s
Medium	62	144	9	0.3s	210s	290s	**1.25**	8min 20s
Large	72	528	33	1.6s	2970s	4165s	**18.3**	33min

(b) Timings for state-wise representation

Instance	λ	ℓ	# of enc. in parallel	Add-RoundKey	Sub-Bytes	Shift-Rows	Mix-Columns	**Total AES** (in hours)	**Relative time**
Small	52	37	37	0.06s	309s	0s	0.09s	**0.74**	1min 12s
Medium	62	138	138	4.5s	3299s	0s	0.44s	**7.86**	3min 25s
Large	72	531	531	27s	47656s	0.04s	2.8s	**113**	12min 46s

Table 2. Timings of byte-wise and state-wise homomorphic AES developed in C++ with GMP, running on a desktop computer (Intel Core i7 at 3.4Ghz, 32GB RAM).

server with 256GB of RAM, while our program runs on a more modest desktop computer with 4 hyper-threaded cores and 32GB of RAM (the whole public key fits in RAM). We claim a slightly lower security level, however: 72 bits versus 80 bits for the implementation from [GHS12b].

Acknowledgments. We would like to thank Taekyoung Kwon and Hyung Tae Lee for valuable comments. The first, third, and fourth authors were supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012-0001243). The fourth author was also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. 2012R1A1A2039129). The last author was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (No. 2011–0025127).

References

[Ben64] Václad E. Beneš. Optimal rearrangeable multistage connecting networks. *Bell Systems Technical Journal*, 43(7):1641–1656, 1964.

[BGH13] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in LWE-based homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2013.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, 2012.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In

Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2009.

[Bih97] Eli Biham. A fast new DES implementation in software. In *FSE '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 97–106. IEEE Computer Society, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.

[CH11] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. Cryptology ePrint Archive, Report 2011/437, 2011. `http://eprint.iacr.org/`.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Batch fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/036, 2013. `http://eprint.iacr.org/`.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2011.

[CN12] Yuanmi Chen and Phong Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 502–519. Springer, 2012.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 446–464. Springer, 2012.

[CT12] Jean-Sébastien Coron and Mehdi Tibouchi. Implementation of the fully homomorphic encryption scheme over the integers with compressed public keys in sage, 2012. `https://github.com/coron/fhe`.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `http://crypto.stanford.edu/craig`.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In Kenneth Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In *CaLC*, pages 51–66, 2001.

[KLYC13] Jinsu Kim, Moon Sung Lee, Aaram Yun, and Jung Hee Cheon. CRT-based fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/057, 2013. `http://eprint.iacr.org/`.

[KS09] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant AES-GCM. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2009.

[Lag85] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM J. Comput.*, 14(1):196–209, 1985.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 1219–1234. ACM, 2012.

[Len87] Jr. Lenstra, H. W. Factoring integers with elliptic curves. *The Annals of Mathematics*, 126(3):pp. 649–673, 1987.

[LP13] Tancrède Lepoint and Pascal Paillier. On the minimal number of bootstrappings in homomorphic circuits. In *WAHC 2013*, Lecture Notes in Computer Science. Springer, 2013. To appear.

[Mem12] Memoirs of the 6th Cryptology Paper Contest, arranged by Korea Communications Commission, 2012.

[NLV11] Michal Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 113–124. ACM, 2011.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[SV11] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations, 2011. *To appear in* Designs, Codes and Cryptography.