Batch Fully Homomorphic Encryption over the Integers*

Jean-Sébastien Coron[1], Tancrède Lepoint[2,3], and Mehdi Tibouchi[4]

[1] Tranef, France
jscoron@tranef.com
[2] CryptoExperts, France
[3] École Normale Supérieure, France
tancrede.lepoint@cryptoexperts.com
[4] NTT Secure Platform Laboratories, Japan
tibouchi.mehdi@lab.ntt.co.jp

Abstract. We extend the fully homomorphic encryption scheme over the integers of van Dijk *et al.* (DGHV) to batch fully homomorphic encryption, *i.e.* to a scheme that supports encrypting and homomorphically processing a vector of plaintext bits as a single ciphertext. Our variant remains semantically secure under the (error-free) approximate-GCD problem. 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: we describe an implementation of the fully homomorphic evaluation of AES encryption, 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, Homomorphic AES.

1 Introduction

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

* An extended abstract [CCK+] will appear at Eurocrypt 2013, merged with some independent but overlapping work from Cheon *et al.*

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 [SV11] as in RLWE-based schemes [BV11a,BV11b], 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 $m = [c \bmod p]_2$. 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 ℓ coprime integers $p_0, \ldots, p_{\ell-1}$. The batch ciphertext has the form

$$c = q \cdot \prod_{i=0}^{\ell-1} p_i + \mathsf{CRT}_{p_0,\ldots,p_{\ell-1}}(2r_0 + m_0, \ldots, 2r_{\ell-1} + m_{\ell-1}),$$

and correctly decrypts to the bit vector (m_i) given by $m_i = [c \bmod p_i]_2$ for all $0 \leqslant i < \ell$.[5] 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 mod 2 of the original plaintexts.

The main challenge, however, is to amend this construction so as to prove semantic security. 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, and using the random $2r$ in (1) to further randomize the ciphertext modulo p.

Extending DGHV public-key encryption to the batch setting may at first seem straightforward: one can use a similar random subset sum technique in the batch variant by generating public key elements x_j with a small residue modulo each of the p_i's instead of only modulo p. However, for the proof of semantic security to go through, the ciphertext c should then be independently randomized modulo each of the p_i's, which isn't easy to achieve without knowing the p_i's. Indeed, if we only use

[5] We denote by $\mathsf{CRT}_{p_0,\ldots,p_{\ell-1}}(a_0, \ldots, a_{\ell-1})$ the unique integer u smaller than $\prod_{i=0}^{\ell-1} p_i$ such that $u \bmod p_i = a_i$ for all $0 \leqslant i < \ell$.

a single additive term $2r$ as in (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 main contribution in this paper is to provide a "correct", provably semantically secure generalization of DGHV to the batch setting. This is done by replacing the term $2r$ in Equation (1) by another subset sum of public key elements which, taken modulo each of the p_i's, generate a lattice with special properties. Our security proof then applies the Leftover Hash Lemma modulo this lattice instead of only modulo q_0. We describe the new batch DGHV scheme in Section 3 and its security proof in Section 3.3. We show that our batch DGHV scheme can encrypt $\ell = \tilde{\mathcal{O}}(\lambda^2)$ bits in a single ciphertext; therefore the ciphertext expansion ratio becomes $\tilde{\mathcal{O}}(\lambda^3)$ instead of $\tilde{\mathcal{O}}(\lambda^5)$ in the original scheme.

In addition to componentwise addition and multiplication, we also show how to perform any permutation on plaintext bits publicly. As opposed to [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 $\mathsf{Recrypt}$ operation 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, with concrete parameters. We use our batch DGHV scheme to homomorphically evaluate the full AES encryption circuit. For the "Large" parameters with $\geqslant 72$ bits of security, our implementation homomorphically encrypts up to 531 AES ciphertexts in parallel in an amortized 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].[6]

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

In this section, we 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 or 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 $-p/2 < [z]_p \leqslant p/2$. For a specific η-bit odd integer p, we use the following distribution over γ-bit integers:

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

$\mathsf{DGHV.KeyGen}(1^\lambda)$**.** 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 $\mathsf{sk} = p$.

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

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

[6] Note that our implementation uses bootstrapping whereas the implementation of [GHS12b] uses leveled homomorphic encryption without bootstrapping.

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 Our Batch DGHV Scheme

We now describe our extension of the DGHV scheme to the batch setting. The goal is to pack ℓ plaintext bits $m_0, \ldots, m_{\ell-1}$ into a single ciphertext. Homomorphic addition and multiplication will then apply in parallel and component-wise on the m_i's.

As explained in the introduction we use Chinese Remaindering with respect to ℓ coprime integers $p_0, \ldots, p_{\ell-1}$ to encrypt $(m_0, \ldots, m_{\ell-1})$ as

$$c = q \cdot \prod_{i=0}^{\ell-1} p_i + \mathsf{CRT}_{p_0,\ldots,p_{\ell-1}}(2r_0 + m_0, \ldots, 2r_{\ell-1} + m_{\ell-1}) \ .$$

By extending the original DGHV public-key encryption equation (2), a plaintext vector $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ could then be encrypted into a single ciphertext:

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + 2r + 2 \sum_{i \in S} x_i \right]_{x_0} \tag{3}$$

where the plaintext elements x_i satisfy $x_i \bmod p_j = r_{i,j}$ and the additional public key elements x'_i are such that $x'_i \bmod p_j = \delta_{i,j} + 2r'_{i,j}$; this yields $[c \bmod p_j]_2 = m_j$ as required.[7]

The main challenge, however, is to obtain a batch DGHV scheme which is still semantically secure. The proof of semantic security for the original DGHV scheme is based on applying the Leftover Hash Lemma on the subset sum, and using the random $2r$ in (1) to further randomize the ciphertext modulo p. However we see that such randomization with $2r$ in (3) does not work in the batch setting, because the *same* random term $2r = 2r \bmod p_i$ is added modulo each of the p_i, 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 semantically secure batch encryption.

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

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

[7] We denote by $\delta_{i,j}$ the Kronecker delta, $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise.

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 (1), 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

$$c = \left[m + 2b \cdot \Pi + 2\sum_{i\in S} x_i \right]_{x_0} \tag{4}$$

for some large random integer b. Modulo p this gives a new integer $r'' = r' + b \cdot \varpi$; 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 $b \cdot \varpi$. 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 (4) 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.

3.1 Description

BDGHV.$\mathsf{KeyGen}(1^\lambda)$. Generate a collection of ℓ random η-bit primes $p_0, \ldots, p_{\ell-1}$, 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[8].

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 $0 \leqslant j < \ell$:

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

Finally, let $\mathsf{pk} = \left\langle x_0, (x_i)_{1\leqslant i\leqslant\tau}, (x'_i)_{0\leqslant i\leqslant\ell-1}, (\Pi_i)_{0\leqslant i\leqslant\ell-1} \right\rangle$ 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 $\boldsymbol{b} = (b_i)_{1\leqslant i\leqslant\tau} \in (-2^\alpha, 2^\alpha)^\tau$ and $\boldsymbol{b'} = (b'_i)_{0\leqslant i\leqslant\ell-1} \in (-2^{\alpha'}, 2^{\alpha'})^\ell$ and output the ciphertext:

$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i=0}^{\ell-1} b'_i \cdot \Pi_i + \sum_{i=1}^{\tau} b_i \cdot x_i \right]_{x_0}. \tag{5}$$

[8] 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.

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

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

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

3.2 Parameters and Correctness

The parameters must be set under the following constraints:

- $\rho \geqslant 2\lambda$ to avoid brute force attack on the noise [CN12],
- $\eta \geqslant \alpha' + \rho' + 1 + \log_2(\ell)$ 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 (see [DGHV10,CMNT11]);
- $\rho' \geqslant \rho + \lambda$ and $\alpha' \geqslant \alpha + \lambda$ for the proof of semantic security (see below),
- $\alpha \cdot \tau \geqslant \gamma + \lambda$ and $\tau \geqslant \ell \cdot (\rho' + 2) + \lambda$ in order to apply the leftover hash lemma (see below).

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], and $\rho' = 3\lambda$, $\alpha' = \tilde{\mathcal{O}}(\lambda^2)$ and $\ell = \tilde{\mathcal{O}}(\lambda^2)$. The main difference with the original DGHV scheme is that the ciphertext expansion ratio becomes $\gamma/\ell = \tilde{\mathcal{O}}(\lambda^3)$ instead of $\gamma = \tilde{\mathcal{O}}(\lambda^5)$. However the public key size (using the compressed public key technique from [CNT12]) becomes $\tilde{\mathcal{O}}(\lambda^7)$ instead of $\tilde{\mathcal{O}}(\lambda^5)$. We refer to Section 4.4 for concrete parameters and timings.

We prove the following lemma in Appendix A. We refer to Appendix A for the definition of correctness for batch homomorphic encryption schemes, with respect to a set $\mathcal{C}_{\mathcal{E}}$ of permitted circuits.

Lemma 1. *The above scheme is correct for $\mathcal{C}_{\mathcal{E}}$.*

3.3 Semantic Security

Ideally we would like to base the security of the new batch DGHV scheme on the same assumption as the original 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 [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 2 (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.*

Theorem 1. *The batch DGHV scheme is semantically secure under the error-free-approximate-GCD assumption.*

3.4 Proof of Theorem 1

We proceed in two steps. First we prove that our batch DGHV scheme is semantically secure under a new assumption, which we call the error-free ℓ-decisional-approximate-GCD assumption. We then show that this new assumption is implied by the (computational) error-free Approximate-GCD assumption from the previous section.

Given integers q_0 and $p_0, \ldots, p_{\ell-1}$, we define the following 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 $r_i \leftarrow (-2^\rho, 2^\rho)$. We denote by $\mathsf{CRT}_{q_0,(p_i)}(q, a_0, \ldots, a_{\ell-1})$ the unique integer u smaller than $x_0 = q_0 \cdot \prod_{i=0}^{\ell-1} p_i$ such that $u \equiv q \ [q_0]$ and $u \equiv a_i \ [p_i]$ for all $0 \leqslant i < \ell$. 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 3 (EF-ℓ-dAGCD$_{\lambda,\gamma,\eta}$). *The error-free 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.*

The decisional problem is therefore to distinguish between an encryption of 0 and an encryption of a random message. To prove semantic security 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.

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

Proof. Under the attack scenario the attacker first receives the public key, and outputs two ℓ-bit messages $\boldsymbol{m}_0$ and $\boldsymbol{m}_1$. The challenger returns an encryption of $\boldsymbol{m}_b$ for a random bit b. The attacker finally outputs a guess b' and succeeds if $b' = b$. We use a sequence of games and denote by S_i the event that the attacker succeeds in $\mathbf{Game}_i$.

$\mathbf{Game}_0$: this is the attack scenario. We simulate the challenger by running KeyGen to obtain pk and sk.

$\mathbf{Game}_1$: we introduce a vector representation of ciphertexts. To any ciphertext $c \bmod x_0$ we associate the vector:

$$\boldsymbol{c} = f(c) = (c \bmod q_0, c \bmod p_0, \ldots, c \bmod p_{\ell-1}) \in \mathbb{Z}_{q_0} \times \mathbb{Z}^\ell.$$

We observe that if a ciphertext c is an encryption of 0 then it is equivalently rewritten as $\boldsymbol{c} = (q_c, 2c_0, \ldots, 2c_{\ell-1}) = \boldsymbol{I_2} \cdot (q_c, c_0, \ldots, c_{\ell-1})$ where $\boldsymbol{I_2}$ is the diagonal matrix with $(1, 2, \ldots, 2)$ on the diagonal.

Given two integers x, y, we have that if $|x \bmod p_i| < p_i/4$ and $|y \bmod p_i| < p_i/4$ for all i, then $f(x+y) = f(x) + f(y)$. Since we only consider ciphertexts which have sufficiently small residues modulo the p_i's, when adding ciphertexts we can therefore work indifferently with integer or our vector representation. In vector representation the encryption equation (5) can then be rewritten as[9]

$$\boldsymbol{c} = (0, \boldsymbol{m})^T + \boldsymbol{I_2} \cdot \left(\boldsymbol{X'} \cdot \boldsymbol{m} + \boldsymbol{X} \cdot \boldsymbol{b} + \boldsymbol{\Pi} \cdot \boldsymbol{b'}\right) \bmod \boldsymbol{x_0}. \tag{6}$$

where the columns of matrices $\boldsymbol{X'}$, $\boldsymbol{X}$ and $\boldsymbol{\Pi}$ contain the vector representations (with the previous $\boldsymbol{I_2}$ factor) of the ciphertexts x'_i (without the $\delta_{i,j}$ terms), x_i and Π_i from the public key:

$$\boldsymbol{X'} = \begin{pmatrix} q_{x'_0} & \cdots & q_{x'_{\ell-1}} \\ r'_{0,0} & \cdots & r'_{\ell-1,0} \\ \vdots & & \vdots \\ r'_{0,\ell-1} & \cdots & r'_{\ell-1,\ell-1} \end{pmatrix}, \ \boldsymbol{X} = \begin{pmatrix} q_{x_1} & \cdots & q_{x_\tau} \\ r_{1,0} & \cdots & r_{\tau,0} \\ \vdots & & \vdots \\ r_{1,\ell-1} & \cdots & r_{\tau,\ell-1} \end{pmatrix}, \ \boldsymbol{\Pi} = \begin{pmatrix} q_{\Pi_0} & \cdots & q_{\Pi_{\ell-1}} \\ \varpi_{0,0} + 2^{\rho'} & \cdots & \varpi_{\ell-1,0} \\ \vdots & \ddots & \vdots \\ \varpi_{0,\ell-1} & \cdots & \varpi_{\ell-1,\ell-1} + 2^{\rho'} \end{pmatrix}.$$

We now modify the encryption Equation (6) as follows. We pre-reduce the term $\boldsymbol{X'} \cdot \boldsymbol{m} + \boldsymbol{X} \cdot \boldsymbol{b}$ modulo the lattice $\boldsymbol{\Pi'}$ formed by the column vectors of $\boldsymbol{x_0}$ and $\boldsymbol{\Pi}$:

$$\boldsymbol{\Pi'} = \begin{pmatrix} q_0 & q_{\Pi_0} & \cdots & q_{\Pi_{\ell-1}} \\ 0 & \varpi_{0,0} + 2^{\rho'} & \cdots & \varpi_{\ell-1,0} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \varpi_{0,\ell-1} & \cdots & \varpi_{\ell-1,\ell-1} + 2^{\rho'} \end{pmatrix},$$

[9] Given a ciphertext c, we denote by $\boldsymbol{c} \bmod \boldsymbol{x_0}$ the vector obtained by reducing the first component modulo q_0.

which gives the new encryption equation:

$$\boldsymbol{c} = (0, \boldsymbol{m})^T + \boldsymbol{I_2} \cdot \left((\boldsymbol{X'} \cdot \boldsymbol{m} + \boldsymbol{X} \cdot \boldsymbol{b}) \bmod \boldsymbol{\Pi'} + \boldsymbol{\Pi} \cdot \boldsymbol{b'} \right) \bmod \boldsymbol{x_0} \ . \tag{7}$$

We prove in Appendix B.3 that two distributions are statistically close, which gives:

Claim 1. $|\Pr[S_1] - \Pr[S_0]| \leqslant \ell \cdot \tau \cdot 2^{\alpha - \alpha' + 2}$.

Game$_2$: we show in Appendix B.4 that using the LHL, the term $\boldsymbol{X} \cdot \boldsymbol{b}$ in Equation (7) is statistically close to uniform modulo $\boldsymbol{\Pi'}$. Therefore we can replace the term $\boldsymbol{X'} \cdot \boldsymbol{m} + \boldsymbol{X} \cdot \boldsymbol{b}$ by a random vector $\boldsymbol{u}$ modulo $\boldsymbol{\Pi'}$ to get

$$\boldsymbol{c} = (0, \boldsymbol{m})^T + \boldsymbol{I_2} \cdot (\boldsymbol{u} + \boldsymbol{\Pi} \cdot \boldsymbol{b'}) \bmod \boldsymbol{x_0}. \tag{8}$$

Claim 2. $|\Pr[S_2] - P[S_1]| \leqslant \ell^2 \cdot \tau \cdot 2^{\rho - \rho' + 2} + \sqrt{2^{\gamma - \alpha \cdot \tau} + \ell \cdot \rho' \cdot 2^{\ell \cdot (\rho' + 2) - \tau}}$.

Game$_3$: we keep the same encryption equation

$$\boldsymbol{c} = (0, \boldsymbol{m})^T + \boldsymbol{I_2} \cdot (\boldsymbol{u} + \boldsymbol{\Pi} \cdot \boldsymbol{b'}) \bmod \boldsymbol{x_0},$$

but we slightly modify the distribution of $\boldsymbol{u}$. Instead of generating a random $\boldsymbol{u}$ modulo $\boldsymbol{\Pi'}$, we generate a vector $\boldsymbol{u}$ whose first component is uniform in $[0, q_0)$ and the other components are uniform in $[0, 2^{\rho'})$. We show in Appendix B.7:

Claim 3. $|\Pr[S_3] - \Pr[S_2]| \leqslant \ell^2 \cdot 2^{\rho - \rho' + 2} + 2^{-\alpha' + 3}$.

Game$_4$: instead of generating the public key with KeyGen, we simulate the public key using the oracle $\mathcal{O}_{q_0,(p_i)}$ from the error-free ℓ-decisional Approximate-GCD problem. Moreover the encryption equation from **Game**$_3$ can be computed directly in integer representation instead of vector representation using $\mathcal{O}_{q_0,(p_i)}$. Therefore we can simulate the public key and ciphertext encryption without knowing the p_i's, and we still have:

$$\Pr[S_4] = \Pr[S_3].$$

Game$_5$: to prepare for the ℓ-decisional problem, we slightly modify the encryption equation with

$$\boldsymbol{c} = (0, \boldsymbol{m})^T + \boldsymbol{I_2} \cdot (\boldsymbol{z} + \boldsymbol{u} + \boldsymbol{\Pi} \cdot \boldsymbol{b'}) \bmod \boldsymbol{x_0}. \tag{9}$$

where $\boldsymbol{z}$ is a vector whose first component is uniform in $[0, q_0)$ and the other components are uniform in $(-2^{\rho}, 2^{\rho})$. From the distribution of $\boldsymbol{z}$ and $\boldsymbol{u}$, we obtain:

$$|\Pr[S_5] - \Pr[S_4]| \leqslant \ell \cdot 2^{\rho - \rho'} \ . \tag{10}$$

Game$_6$: we modify again the encryption equation by adding a vector $\boldsymbol{z_1} = (0, \boldsymbol{v_1})^T$ where $\boldsymbol{v_1} \leftarrow \{0,1\}^{\ell}$, as follows

$$\boldsymbol{c} = (0, \boldsymbol{m})^T + \boldsymbol{z_1} + \boldsymbol{I_2} \cdot (\boldsymbol{z} + \boldsymbol{u} + \boldsymbol{\Pi} \cdot \boldsymbol{b'}) \bmod \boldsymbol{x_0}. \tag{11}$$

It is easy to see that the gap between Game 5 and Game 6 is at most the distinguishing advantage of an efficient adversary against the error-free ℓ-decisional-approximate-GCD problem. Namely the vector $\boldsymbol{I_2} \cdot \boldsymbol{z}$ in Equation (9) can be obtained through $z = \mathcal{O}_{q_0,(p_i)}(\boldsymbol{v}_0)$ where $\boldsymbol{v}_0 = 0^{\ell}$, whereas the vector $\boldsymbol{z_1} + \boldsymbol{I_2} \cdot \boldsymbol{z}$ can be obtained from $z = \mathcal{O}_{q_0,(p_i)}(\boldsymbol{v}_1)$ where $\boldsymbol{v_1} \leftarrow \{0,1\}^{\ell}$. Therefore

$$|\Pr[S_6] - \Pr[S_5]| \leqslant \varepsilon_{\text{EF-}\ell\text{-dAGCD}}.$$

Game$_7$: we remove the term $(0, \boldsymbol{m})^T$ in (11). Then the adversary's view in Game 7 is independent from $\boldsymbol{m}$ and we get

$$\Pr[S_7] = \frac{1}{2}.$$

The statistical distance between the random variables $(0, \boldsymbol{m})^T + \boldsymbol{z_1} + \boldsymbol{I_2} \cdot \boldsymbol{z}$ and $\boldsymbol{z_1} + \boldsymbol{I_2} \cdot \boldsymbol{z}$ is at most $\ell \cdot 2^{-\rho}$. Therefore:

$$|\Pr[S_7] - \Pr[S_6]| \leqslant \ell \cdot 2^{-\rho}.$$

Finally all the previous probability gaps can be made negligible by satisfying the constraints on the parameters from Section 3.2; this concludes the proof of Lemma 2. □

We then prove the following Lemma in Appendix C. Combined with Lemma 2 this proves Theorem 1.

Lemma 3. *The ℓ-decisional-Approximate-GCD problem is hard if the error-free-approximate-GCD problem is hard.*

4 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.

4.1 The Squashed Scheme

As mentioned in the introduction, to follow Gentry's blueprint and make our somewhat homomorphic scheme 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:

BDGHV.KeyGen(1^λ). 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

$$\frac{1}{p_j} = \sum_{i=0}^{\Theta-1} s_{j,i} \cdot y_i + \varepsilon_j \bmod 2 \tag{12}$$

for some $|\varepsilon_j| < 2^{-\kappa}$.

Output the secret key $\mathsf{sk} = (\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{\ell-1})$ and public key $\mathsf{pk} = (\mathsf{pk}^*, y_0, \ldots, y_{\Theta-1})$.

BDGHV.Expand(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 $n = \lceil \log_2(\theta + 1) \rceil$ 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.Decrypt($\mathsf{sk}, c, \boldsymbol{z}$). Output $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ with

$$m_j \leftarrow \left[\left\lfloor \sum_{i=0}^{\Theta-1} s_{j,i} \cdot z_i \right\rfloor \right]_2 \oplus (c \bmod 2). \tag{13}$$

This completes the description of the scheme. We use $n = \lceil \log_2(\theta + 1) \rceil$ as in [CMNT11]; the proof of the following Lemma is the same as in [CMNT11, Appendix E].

Lemma 4. *The BDGHV encryption scheme is correct for the set $C(\mathcal{P}_\mathcal{E})$ of circuits that compute permitted polynomials.*

Remark 1. To reduce the size of the public key we can generate all the y_i's pseudo-randomly as in [CMNT11], except ℓ of them in order to satisfy Equation (12) for all $0 \leqslant j < \ell$.

4.2 Bootstrapping

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

In the original DGHV scheme, the decryption equation was:

$$m \leftarrow \left[\left\lfloor \sum_{i=0}^{\Theta-1} s_i \cdot z_i \right\rceil\right]_2 \oplus (c \bmod 2) \tag{14}$$

and could be homomorphically evaluated by providing an encryption σ_i of every secret-key bit s_i; one would obtain a new ciphertext which would encrypt the same plaintext bit m but with a possibly reduced noise.

Similarly, the decryption Equation (13) 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:

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

This gives a new ciphertext that encrypts the same ℓ-bit plaintext vector, but with a (possibly) reduced noise. In other words, instead of having an homomorphic evaluation of a single Equation (14), we have that the ℓ equations in (13) are homomorphically evaluated in parallel, one in each of the ℓ plaintext slots of the ciphertext. Therefore the Recrypt operation is done in parallel over the ℓ slots, with the same complexity as a single Recrypt operation in the original scheme.

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 Theorem 5.1 in [CMNT11].

Theorem 2. *Let $\mathcal{E}$ be the above scheme, and let $D_\mathcal{E}$ be the set of augmented (squashed) decryption circuits. Then $D_\mathcal{E} \subset C(\mathcal{P}_\mathcal{E})$.*

4.3 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. 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 described independently in [BGH12] 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

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

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

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

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 $\mathfrak{S}_\ell$, 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 tradeoff. 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.

4.4 Implementation Results

We provide in Table 1 concrete key sizes and timings for our 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]. We provide a complete description of the scheme in Appendix D. 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	$\gamma \times 10^{-6}$	τ	Θ	pk size
Toy	42	10	26	988	0.29	188	150	647kB
Small	52	37	41	1558	1.6	661	555	13.3MB
Medium	62	138	56	2128	8.5	2410	2070	304MB
Large	72	531	71	2698	39	8713	7965	5.6GB

KeyGen	Encrypt	Decrypt	Mult	Expand	Recrypt
0.06s	0.02s	0s	0.003s	0.007s	0.11s
1.74s	0.23s	0.02s	0.025s	0.08s	1.10s
73s	3.67s	0.45s	0.16s	1.60s	11.9s
3493s	61s	9.8s	0.72s	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).

5 Homomorphic Evaluation of the AES Circuit

In this section, we show how to homomorphically evaluate the AES-128 encryption circuit using our batch encryption scheme, and provide concrete timings. A similar implementation with the RLWE-based fully-homomorphic encryption scheme [BV11a,BV11b,GHS12a] is 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 send encrypted under AES with a ciphertext expansion equal to 1, along with the public key $\mathsf{pk}_{\mathsf{FHE}}$ of the FHE scheme 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 $\mathsf{pk}_{\mathsf{FHE}}$.[10]

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 requires 16 slots of 8 ciphertexts, and one can perform $k = \ell/16$ AES encryptions in parallel using these 8 ciphertexts. Formally the AES state is composed of 8 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.[11] We briefly describe how to implement the AES stages; more details are provided in Appendix E.

The AddRoundKey stage performs a XOR between the AES state and the current round key. This operation only consists of 8 BDGHV.Add operations. The SubBytes stage is implemented using the 115 gates circuit of Boyar and Peralta [BP10] to compute the Sbox. To minimize the number of Recrypt, we perform the Recrypt operation only 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; it requires 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 BDGHV.Add, 320 BDGHV.Mult, and 377 BDGHV.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 BDGHV.Add, 5120 BDGHV.Mult and 2448 BDGHV.Recrypt. More details are provided in Appendix E.

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 server with 256GB of RAM, while our program runs on a more modest desktop computer with 4 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].

[10] Note that we focus here on AES encryption rather than AES decryption to be consistent with [GHS12b].

[11] 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].

(a) Timings for byte-wise representation

Instance	λ	ℓ	# of enc. in parallel	AddRoundKey	ShiftRows and SubBytes	MixColumns	**Total AES** (in hours)	**Relative time**
Toy	42	16	1	0.006s	2.2s	3s	**0.013**	48s
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	AddRoundKey	SubBytes	ShiftRows	MixColumns	**Total AES** (in hours)	**Relative time**
Toy	42	10	10	0.06s	33s	0s	0.02s	**0.08**	29s
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).

References

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

[BGH12] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in LWE-based homomorphic encryption. Cryptology ePrint Archive, Report 2012/565, 2012. `http://eprint.iacr.org/`. *To appear in PKC 2013.*

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

[Bih97] Eli Biham. A fast new DES implementation in software. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, 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, *Advances in Cryptology – 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, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.

[CCK+] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *To Appear at Eurocrypt 2013.*

[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, *Advances in Cryptology – 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, *Advances in Cryptology – 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, *Advances in Cryptology – 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, *Advances in Cryptology – 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. `crypto.stanford.edu/craig`.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In Kenneth Paterson, editor, *Advances in Cryptology – 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, *Advances in Cryptology – 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.

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

[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.

[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.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Nguyen and David Pointcheval, editors, *Public Key Cryptography – 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.

A Correctness of the Scheme

We recall and adapt to the batch settings the definition of correctness from [Gen09,DGHV10]. We consider an homomorphic public-key encryption scheme $\mathcal{E}$ with an additional algorithm $\mathsf{Evaluate}$ taking as input the public key pk, a mod-2 arithmetic circuit C with t inputs and t ciphertexts c_i, and outputting another ciphertext c.

Definition 4 (Correct batch homomorphic decryption). *The scheme*

$$\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$$

is correct for a given t-input circuit C if, for any key-pair $(\mathsf{sk}, \mathsf{pk})$ output by $\mathsf{KeyGen}(\lambda)$, any t plaintext ℓ-bit vectors $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_t$, and any ciphertexts $\boldsymbol{C} = (c_1, \ldots, c_t)$ with $c_i \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \boldsymbol{m}_i)$, it holds that

$$\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Evaluate}(\mathsf{pk}, C, \boldsymbol{C})) = \Big(C(\boldsymbol{m}_1[0], \ldots, \boldsymbol{m}_t[0]), \ldots, C(\boldsymbol{m}_1[\ell-1], \ldots, \boldsymbol{m}_t[\ell-1])\Big).$$

As in [Gen09,DGHV10], we define a *permitted circuit* as one where for any $i \geqslant 1$ and any set of integers inputs less that $\ell^i 2^{i(\alpha'+\rho'+2)}$ in absolute value, the generalized circuit's output has absolute value at most $2^{i(\eta-3-n)}$ with $n = \lceil \log_2(\lambda+1) \rceil$; we let $\mathcal{C}_{\mathcal{E}}$ be the set of permitted circuits.

We show that the batch scheme presented in Section 3 is correct.

Proof (of Lemma 1). Given a ciphertext c outputted by $\mathsf{BDGHV.Encrypt}(\mathsf{pk}, \boldsymbol{m})$, there exist integer vectors $\boldsymbol{b} = (b_i)_{1 \leqslant i \leqslant \tau} \in (-2^\alpha, 2^\alpha)^\tau$ and $\boldsymbol{b'} = (b'_i)_{1 \leqslant i \leqslant \ell-1} \in (-2^{\alpha'}, 2^{\alpha'})$ such that

$$c = \sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i=1}^{\tau} b_i \cdot x_i + \sum_{i=0}^{\ell-1} b'_i \cdot \Pi_i \mod x_0$$

For each $j = 0, \ldots, \ell-1$, this gives

$$|c \bmod p_j| \leqslant \ell \cdot 2^{\rho+1} + \tau 2^{\alpha+\rho'+1} + \ell \cdot 2^{\alpha'+\rho'+1} \leqslant \ell \cdot 2^{\alpha'+\rho'+2}. \tag{15}$$

Let C be a permitted circuit with t inputs and let $C^\dagger$ be the corresponding circuit operations over the integers rather than modulo 2. Let $c_i \leftarrow \mathsf{BDGHV.Encrypt}(\mathsf{pk}, \boldsymbol{m}_i)$. We have, for each $j = 0, \ldots, \ell - 1$,

$$c \bmod p_j = C^\dagger(c_1, \ldots, c_t) \bmod p_j = C^\dagger(c_1 \bmod p_j, \ldots, c_t \bmod p_j) \bmod p_j. \tag{16}$$

From (15) and the definition of permitted circuits, we obtain

$$\left| C^\dagger(c_1 \bmod p_j, \ldots, c_t \bmod p_j) \right| \leqslant 2^{\eta-4} \leqslant p_j/8.$$

Therefore

$$C^\dagger(c_1 \bmod p_j, \ldots, c_t \bmod p_j) \bmod p_j = C^\dagger(c_1 \bmod p_j, \ldots, c_t \bmod p_j),$$

which implies from (16) that $c \bmod p_j = C^\dagger(c_1 \bmod p_j, \ldots, c_t \bmod p_j)$, and eventually

$$[c \bmod p_j]_2 = \left[C^\dagger([c_1 \bmod p_j]_2, \ldots, [c_t \bmod p_j]_2) \right]_2 = C(\boldsymbol{m}_1[j], \ldots, \boldsymbol{m}_t[j]),$$

which concludes the proof. □

B Proof of Lemma 2

In Sections B.1 and B.2, we establish some preliminary results.

B.1 Cardinality of the Kernel of a Linear Map

Lemma 5. *Let G be a finite abelian group of order $|G|$. Let $t \geqslant 1$ be an integer. For any $\boldsymbol{a} = (a_1, \ldots, a_t) \in \mathbb{Z}^t$, define the linear map $\phi_{\boldsymbol{a}} \colon G^t \to G$ by $\phi_{\boldsymbol{a}}(g_1, \ldots, g_t) = \sum_{i=1}^t a_i \cdot g_i$. The cardinality of the kernel of $\phi_{\boldsymbol{a}}$ is given by $|\ker \phi_{\boldsymbol{a}}| = |G|^{t-1} \cdot \prod_{j=1}^r \gcd(d_j, a_1, \ldots, a_t)$, where the d_j's are the invariant factors of G, i.e. satisfy $d_j \mid d_{j+1}$ and $G \simeq \prod_{j=1}^r \mathbb{Z}/d_j\mathbb{Z}$.*

Proof. We prove this result componentwise. Let $\boldsymbol{a} \in \mathbb{Z}^t$. For any index $j \in [1, r]$, consider the map

$$\begin{aligned} \phi_{j,\boldsymbol{a}} \colon (\mathbb{Z}/d_j\mathbb{Z})^t &\to \mathbb{Z}/d_j\mathbb{Z} \\ (g_1, \ldots, g_t) &\mapsto \sum_{i=1}^t a_i \cdot g_i. \end{aligned}$$

Its image is the subgroup of $\mathbb{Z}/d_j\mathbb{Z}$ generated by $\gcd(a_1, \ldots, a_t)$, hence

$$|\mathrm{im}(\phi_{j,\boldsymbol{a}})| = d_j / \gcd(d_j, a_1, \ldots, a_t),$$

which implies

$$|\ker \phi_{j,\boldsymbol{a}}| = d_j^t / |\mathrm{im}(\phi_{j,\boldsymbol{a}})| = d_j^{t-1} \cdot \gcd(d_j, a_1, \ldots, a_t).$$

The result follows directly. □

B.2 Diagonally Dominant Matrices

Given a matrix $B = (b_{ij}) \in \mathbb{Z}^{n \times n}$, we let $\Lambda_i(B) = \sum_{k \neq i} |b_{ik}|$.

Definition 5 (Diagonally dominant matrix). *A matrix $B = (b_{ij}) \in \mathbb{Z}^{n \times n}$ is said to be* diagonally dominant *if $|b_{ii}| > \Lambda_i(B)$ for all i.*

We first show that a diagonally dominant matrix is invertible and give a bound on the operator norm of its inverse.

Lemma 6. *Let $B = (b_{ij}) \in \mathbb{Z}^{n\times n}$ be a diagonally dominant matrix. Then B is invertible and*

$$\|B^{-1}\|_\infty \leqslant \max_{i=1,\dots,n} \left(|b_{ii}| - \Lambda_i(B)\right)^{-1}.$$

where $\|\cdot\|_\infty$ is the operator norm on $n \times n$ matrices with respect to the ℓ^∞ norm on $\mathbb{R}^n$.

Proof. For any vector $\boldsymbol{t} \in \mathbb{R}^n$, we have

$$\|B \cdot \boldsymbol{t}\|_\infty = \max_{i=1,\dots,n} \left| \sum_{j=1}^{n} b_{ij} \cdot t_j \right|$$

There exists k such that $\|\boldsymbol{t}\|_\infty = |t_k|$. Consequently

$$\|B \cdot \boldsymbol{t}\|_\infty \geqslant \left| \sum_{j=1}^{n} b_{kj} \cdot t_j \right| \geqslant |t_k| \cdot (|b_{kk}| - \Lambda_k(B)) \geqslant \|\boldsymbol{t}\|_\infty \cdot \min_{i=1,\dots,n} (|b_{ii}| - \Lambda_i(B)) \geqslant \|\boldsymbol{t}\|_\infty \cdot \varepsilon. \quad (17)$$

for $\varepsilon := \min_{i=1,\dots,n} (|b_{ii}| - \Lambda_i(B))$, where $\varepsilon > 0$ since B is a diagonally dominant matrix.

This implies that the matrix B is invertible, since $B\boldsymbol{t} = \boldsymbol{0} \Rightarrow \|B\boldsymbol{t}\|_\infty = 0 \Rightarrow \|\boldsymbol{t}\|_\infty = 0 \Rightarrow \boldsymbol{t} = \boldsymbol{0}$. Given $\boldsymbol{u} \in \mathbb{R}^n$, let $\boldsymbol{t} = B^{-1}\boldsymbol{u}$. Then from Equation (17) with $B\boldsymbol{t} = \boldsymbol{u}$

$$\|\boldsymbol{u}\|_\infty \geqslant \|B^{-1}\boldsymbol{u}\|_\infty \cdot \varepsilon$$

Thus

$$\|B^{-1}\|_\infty \leqslant \max_{i=1,\dots,n} \left(|b_{ii}| - \Lambda_i(B)\right)^{-1}.$$

□

Lemma 7. *Let $B = (b_{ij}) \in \mathbb{Z}^{n\times n}$ be a diagonally dominant matrix, with $b_{ii} > 0$ for all i. Let U be the set of vectors $(u_i) \in \mathbb{Z}^n$ such that $|u_i| < \Delta_i/2$ where $\Delta_i = b_{ii} - \Lambda_i(B)$ for all i. The vectors in U are all distinct modulo B.*

Proof. Let $\boldsymbol{u}, \boldsymbol{u'}$ be two distinct vectors in U. We have $|u_i - u'_i| < b_{ii} - \Lambda_i(B)$ for all i. Let $\boldsymbol{v}$ be a non-zero vector in the lattice generated by the column vectors of B; we show that $|v_i| \geqslant b_{ii} - \Lambda_i(B)$ for some i. Therefore we must have $\boldsymbol{u} \neq \boldsymbol{u'} \bmod B$.

We write $\boldsymbol{v} = B \cdot \boldsymbol{y}$ for some non-zero vector $\boldsymbol{y} \in \mathbb{Z}^n$. Let $M = \max\{|y_k|; 1 \leqslant k \leqslant n\}$ and let i such that $|y_i| = M$. We write:

$$v_i = \sum_{j=1}^{n} b_{ij} y_j = b_{ii} y_i + \sum_{j\neq i} b_{ij} y_j$$

which gives:

$$|b_{ii} y_i| = |v_i - \sum_{j\neq i} b_{ij} y_j| \leqslant |v_i| + \sum_{j\neq i} |b_{ij}| \cdot M \leqslant |v_i| + M \cdot \Lambda_i(B)$$

which gives using $M \geqslant 1$:

$$|v_i| \geqslant |b_{ii} y_i| - M \cdot \Lambda_i(B) \geqslant M \cdot (b_{ii} - \Lambda_i(B)) \geqslant b_{ii} - \Lambda_i(B).$$

□

B.3 Proof of Claim 1

Given an invertible matrix $\boldsymbol{A} \in \mathbb{Z}^{n\times n}$ and a vector $\boldsymbol{y} \in \mathbb{Z}^n$, we denote by $\boldsymbol{x} = \boldsymbol{y} \bmod \boldsymbol{A}$ the vector $\boldsymbol{x} = \boldsymbol{y} - \boldsymbol{A} \cdot \lfloor \boldsymbol{A}^{-1} \cdot \boldsymbol{y} \rfloor$.

Since the first component of $\boldsymbol{c}$ is always reduced modulo q_0, we can restrict ourselves to the ℓ last components of the vectors, and to the sub-matrix

$$\boldsymbol{\Pi''} = \begin{pmatrix} \varpi_{0,0} + 2^{\rho'} & \cdots & \varpi_{\ell-1,0} \\ \vdots & \ddots & \vdots \\ \varpi_{0,\ell-1} & \cdots & \varpi_{\ell-1,\ell-1} + 2^{\rho'} \end{pmatrix} .$$

More precisely, letting $\boldsymbol{u} := \boldsymbol{X'} \cdot \boldsymbol{m} + \boldsymbol{X} \cdot \boldsymbol{b}$, we have

$$\begin{aligned} (\boldsymbol{u} \bmod \boldsymbol{\Pi'}) \bmod \boldsymbol{x_0} &= \boldsymbol{u} - \boldsymbol{\Pi'} \cdot \lfloor \boldsymbol{\Pi'}^{-1} \cdot \boldsymbol{u} \rfloor \bmod \boldsymbol{x_0} \\ &= \boldsymbol{u} - \boldsymbol{\Pi} \cdot \lfloor \boldsymbol{\Pi''}^{-1} \cdot \boldsymbol{u'} \rfloor \bmod \boldsymbol{x_0} . \end{aligned}$$

where $\boldsymbol{u'} \in \mathbb{Z}^\ell$ is the vector $\boldsymbol{u}$ with its first component removed. This gives

$$\boldsymbol{c} = (0, \boldsymbol{m})^T + \boldsymbol{I_2} \cdot \big(\boldsymbol{u} + \boldsymbol{\Pi} \cdot \big(\boldsymbol{b'} - \lfloor \boldsymbol{\Pi''}^{-1} \cdot \boldsymbol{u'} \rfloor\big)\big) \bmod \boldsymbol{x_0} .$$

We have

$$\|\boldsymbol{u'}\|_\infty \leqslant \ell \cdot 2^\rho + \tau \cdot 2^{\rho'+\alpha} \leqslant \tau \cdot 2^{\rho'+\alpha+1} .$$

Moreover since $\boldsymbol{\Pi''}$ is a diagonally-dominant matrix, we have from Lemma 6 in Appendix B.2

$$\|\boldsymbol{\Pi''}^{-1}\|_\infty \leqslant \max_{i=0,\ldots,\ell-1} \left(\left| \varpi_{i,i} + 2^{\rho'} \right| - \sum_{k\neq i} |\varpi_{k,i}| \right)^{-1} \leqslant 2^{-\rho'+1} .$$

Therefore

$$\|\lfloor \boldsymbol{\Pi''}^{-1} \cdot \boldsymbol{u'} \rfloor\|_\infty \leqslant \|\boldsymbol{\Pi''}^{-1} \cdot \boldsymbol{u'}\|_\infty + 1 \leqslant \|\boldsymbol{\Pi''}^{-1}\|_\infty \cdot \|\boldsymbol{u'}\|_\infty + 1 \leqslant \tau \cdot 2^{\alpha+3} .$$

Since the vector $\boldsymbol{b'}$ has its ℓ components uniformly distributed in $(-2^{\alpha'}, 2^{\alpha'})$, for any fixed vector $\lfloor \boldsymbol{\Pi''}^{-1} \cdot \boldsymbol{u'} \rfloor$ the statistical distance between $\boldsymbol{b'}$ and $\boldsymbol{b'} - \lfloor \boldsymbol{\Pi''}^{-1} \cdot \boldsymbol{u'} \rfloor$ is at most $\ell \cdot \tau \cdot 2^{\alpha+2-\alpha'}$; this gives the same upper-bound for the statistical distance between the ciphertext $\boldsymbol{c}$ in Game_0 and Game_1, which proves the claim.

B.4 Proof of Claim 2

Our goal is to show that the distribution of the term $\boldsymbol{X} \cdot \boldsymbol{b} \bmod \boldsymbol{\Pi'}$ is statistically close to the uniform distribution modulo $\boldsymbol{\Pi'}$. First, we show that the distribution of $\boldsymbol{X} \bmod \boldsymbol{\Pi'}$ is statistically close to the uniform distribution of matrices with τ columns in $Y = \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'})$.

Lemma 8. *The distribution of $\boldsymbol{X}$ is ε-statistically close to uniform over Y^τ, where*

$$\varepsilon = \ell^2 \cdot \tau \cdot 2^{\rho-\rho'+2} .$$

Therefore instead of considering the distribution of $\boldsymbol{X} \cdot \boldsymbol{b} \bmod \boldsymbol{\Pi'}$ we can consider the distribution of $\boldsymbol{G} \cdot \boldsymbol{b} \bmod \boldsymbol{\Pi'}$ where $\boldsymbol{G}$ is uniformly distributed over the matrices with τ columns in $Y = \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'})$. For this we consider the family $\mathcal{H}$ of hash functions $h_{\boldsymbol{G}} \colon X \to Y$ where $X = (-2^\alpha, 2^\alpha)^\tau$ and $Y = \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'})$ defined as

$$h_{\boldsymbol{G}}(\boldsymbol{b}) = \boldsymbol{G} \cdot \boldsymbol{b} \bmod \boldsymbol{\Pi'},$$

where $\boldsymbol{G}$ is a matrix of τ columns vectors over $Y = \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'})$. Using the Leftover Hash Lemma, we show:

Lemma 9. *The distribution of $(\boldsymbol{G}, \boldsymbol{G} \cdot \boldsymbol{b} \bmod \boldsymbol{\Pi'})$ is ε_2-statistically close to uniform modulo $\boldsymbol{\Pi'}$, with*

$$\varepsilon_2 = \sqrt{2^{\gamma-\alpha\cdot\tau} + \ell \cdot \rho' \cdot 2^{\ell\cdot(\rho'+2)-\tau}}$$

Combining Lemma 8 and Lemma 9 this proves Claim 2.

B.5 Proof of Lemma 8

We must show that the distribution of the vectors $\boldsymbol{x}_i$'s is statistically close to uniform modulo $\boldsymbol{\Pi'}$. For this it suffices to show that the distribution of the vectors $\boldsymbol{r}_i = (r_{i,0}, \ldots, r_{i,\ell-1}) \in \mathbb{Z}^\ell$ is statistically close to uniform modulo $\boldsymbol{\Pi''}$, where

$$\boldsymbol{\Pi''} = \begin{pmatrix} \varpi_{0,0} + 2^{\rho'} & \cdots & \varpi_{\ell-1,0} \\ \vdots & \ddots & \vdots \\ \varpi_{0,\ell-1} & \cdots & \varpi_{\ell-1,\ell-1} + 2^{\rho'} \end{pmatrix} .$$

Namely the first coefficient of the $\boldsymbol{x}_i$'s is uniform modulo q_0 and independent of the other coefficients, therefore it is distributed as the first coefficient of a random vector modulo $\boldsymbol{\Pi'}$.

The vectors $\boldsymbol{r}_i = (r_{i,0}, \ldots, r_{i,\ell-1}) \in \mathbb{Z}^\ell$ are uniformly and independently distributed in the set

$$U_0 = \{(u_j)_{j=0,\ldots,\ell-1} \in \mathbb{Z}^\ell : -2^{\rho'-1} < u_j < 2^{\rho'-1}\} .$$

We must show that when $\boldsymbol{r} \leftarrow U_0$ the distribution of $\boldsymbol{r} \bmod \boldsymbol{\Pi''}$ is statistically close to uniform in $\mathbb{Z}^\ell/(\boldsymbol{\Pi''})$. We consider the following subset of U_0:

$$U_0' = \{(u_j)_{j=0,\ldots,\ell-1} \in \mathbb{Z}^\ell : |u_j| < \Delta_j/2\}$$

where $\Delta_j := 2^{\rho'} - 2^\rho - \Lambda_j(\boldsymbol{\Pi''})$. From Lemma 7 we have that the vectors of U_0' are all distinct modulo $\boldsymbol{\Pi''}$. Therefore when $\boldsymbol{r} \leftarrow U_0'$ (instead of U_0) then $\boldsymbol{r} \bmod \boldsymbol{\Pi''}$ is uniformly distributed in the set:

$$U_1 = \{\boldsymbol{u} \bmod \boldsymbol{\Pi''} : \boldsymbol{u} \in U_0'\} \subset \mathbb{Z}^\ell/(\boldsymbol{\Pi''})$$

To prove Lemma 8 it suffices to show that the sets U_0 and U_0' have very close size, and that the sets U_1 and $\mathbb{Z}^\ell/(\boldsymbol{\Pi''})$ have very close size. We know that $|U_0'| \leqslant |U_0| \leqslant 2^{\rho' \cdot \ell}$ and that $|U_0'| = |U_1| \leqslant |\mathbb{Z}^\ell/(\boldsymbol{\Pi''})| = |\det \boldsymbol{\Pi''}|$; therefore it suffices to obtain a lower-bound for $|U_0'|$ and an upper-bound for $|\det \boldsymbol{\Pi''}|$.

To obtain a lower-bound for $|U_0'|$ we write:

$$|U_0| \geqslant |U_0'| \geqslant \prod_{j=0}^{\ell-1} \Delta_j \geqslant (2^{\rho'} - \ell \cdot 2^\rho)^\ell \geqslant 2^{\rho' \cdot \ell} \cdot \left(1 - \ell^2 \cdot 2^{\rho - \rho'}\right) . \tag{18}$$

To upper bound $|\det \boldsymbol{\Pi''}|$ we use Hadamard's inequality

$$\left|\det \boldsymbol{\Pi''}\right| \leqslant \prod_{j=0}^{\ell-1} \|\boldsymbol{\pi}_j'\|,$$

where $\boldsymbol{\pi}_j'$ is the j-th row of $\boldsymbol{\Pi''}$. We have for all j:

$$\begin{aligned} \|\boldsymbol{\pi}_j'\| &\leqslant \sqrt{(2^{\rho'} + 2^\rho)^2 + (\ell - 1) \cdot (2^\rho)^2} \leqslant \sqrt{2^{2\rho'} + 2^{\rho' + \rho + 1} + \ell \cdot 2^{2\rho}} \\ &\leqslant \sqrt{2^{2\rho'}(1 + 2^{\rho + 2 - \rho'})} \leqslant 2^{\rho'} \cdot \left(1 + 2^{\rho + 1 - \rho'}\right) . \end{aligned}$$

Thus

$$\left|\det(\boldsymbol{\Pi''})\right| \leqslant 2^{\rho' \cdot \ell} \cdot \left(1 + \ell \cdot 2^{\rho + 1 - \rho'}\right)$$

Combining the previous inequality with (18), we obtain:

$$2^{\rho' \cdot \ell} \cdot \left(1 - \ell^2 \cdot 2^{\rho - \rho'}\right) \leqslant |U_0'| = |U_1| \leqslant |\mathbb{Z}^\ell/(\boldsymbol{\Pi''})| \leqslant 2^{\rho' \cdot \ell} \cdot \left(1 + \ell \cdot 2^{\rho + 1 - \rho'}\right) . \tag{19}$$

Combining inequalities (18) and (19) we obtain that the statistical distance between $\boldsymbol{r} \bmod \boldsymbol{\Pi''}$ with $\boldsymbol{r} \leftarrow U_0$ and the uniform distribution in $\mathbb{Z}^\ell/(\boldsymbol{\Pi''})$ is at most $\ell^2 \cdot 2^{\rho - \rho' + 2}$, which proves Lemma 8.

B.6 Proof of Lemma 9

We consider the family $\mathcal{H}$ of hash functions $h_{\boldsymbol{G}}\colon X \to Y$ where $X = (-2^\alpha, 2^\alpha)^\tau$ and $Y = \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'})$ defined as

$$h_{\boldsymbol{G}}(\boldsymbol{b}) = \boldsymbol{G} \cdot \boldsymbol{b} \bmod \boldsymbol{\Pi'},$$

where $\boldsymbol{G}$ is a matrix of τ columns vectors over $Y = \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'})$. We must show that $(h_{\boldsymbol{G}}, h_{\boldsymbol{G}}(\boldsymbol{b}))$ is uniformly distributed in $\mathcal{H} \times Y$ when $h \leftarrow \mathcal{H}$ and $\boldsymbol{b} \leftarrow X$.

We first recall the notion of ε-pairwise independence introduced in [CMNT11].

Definition 6. *A family $\mathcal{H}$ of hash functions $h\colon X \to Y$ is ε-pairwise independent if*

$$\sum_{x \neq x'} \left(\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] - \frac{1}{|Y|} \right) \leqslant |X|^2 \cdot \frac{\varepsilon}{|Y|}.$$

The following generalization of the usual leftover hash lemma is proved in [CMNT11].

Lemma 10 (Leftover hash lemma). *Let $\mathcal{H}$ be a family of ε-pairwise independent hash functions. Suppose that $h \leftarrow \mathcal{H}$ and $x \leftarrow X$ are chosen uniformly and independently. Then $(h, h(x))$ is $(\frac{1}{2}\sqrt{|Y|/|X| + \varepsilon})$-uniform over $\mathcal{H} \times Y$.*

Therefore to apply Lemma 10 we show that the hash function family $h_{\boldsymbol{G}}(\boldsymbol{b}) = \boldsymbol{G} \cdot \boldsymbol{b} \bmod \boldsymbol{\Pi'}$ is ε-pairwise independent. We must bound the value δ defined by:

$$\delta = \frac{|Y|}{|X|^2} \sum_{\boldsymbol{b} \neq \boldsymbol{b'}} \left(\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] - \frac{1}{|Y|} \right).$$

Let $\boldsymbol{b}, \boldsymbol{b'} \in X$, $\boldsymbol{b} \neq \boldsymbol{b'}$ and define $\boldsymbol{a} = \boldsymbol{b} - \boldsymbol{b'}$. We denote by $(q_i, \boldsymbol{r_i}) \in \mathbb{Z}^{\ell+1}$ the column vectors of $\boldsymbol{G}$, for $1 \leqslant i \leqslant \tau$. We have that

$$h(\boldsymbol{a}) = \boldsymbol{0} \in \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'}) \iff \begin{cases} \phi_{\boldsymbol{a},q_0}\big((q_i)_{i=1,\ldots,\tau}\big) & := \sum_{i=1}^{\tau} a_i \cdot q_i = 0 \bmod q_0 \\ \phi_{\boldsymbol{a},\boldsymbol{\Pi''}}\big((\boldsymbol{r}_i)_{i=1,\ldots,\tau}\big) & := \sum_{i=1}^{\tau} a_i \cdot \boldsymbol{r}_i = 0 \bmod \boldsymbol{\Pi''} \end{cases},$$

where

$$\boldsymbol{\Pi''} = \begin{pmatrix} \varpi_{0,0} + 2^{\rho'} & \cdots & \varpi_{\ell-1,0} \\ \vdots & \ddots & \vdots \\ \varpi_{0,\ell-1} & \cdots & \varpi_{\ell-1,\ell-1} + 2^{\rho'} \end{pmatrix}.$$

Therefore for any fixed $\boldsymbol{b} \neq \boldsymbol{b'}$

$$\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] = \frac{|\ker \phi_{\boldsymbol{a},q_0}|}{q_0^\tau} \cdot \frac{\left|\ker \phi_{\boldsymbol{a},\boldsymbol{\Pi''}}\right|}{|\det(\boldsymbol{\Pi''})|^\tau}.$$

We have that $\gcd(a_1, \ldots, a_\tau, q_0) = 1$. Indeed, all the factors of q_0 are such that $2^{\lambda^2} > 2^{\alpha+1} > a_j$ for all j, and thus any nonzero a_j is coprime with q_0. Consequently, applying Lemma 5 to $G = \mathbb{Z}_{q_0}$, we get

$$\frac{|\ker \phi_{\boldsymbol{a},q_0}|}{q_0^\tau} = \frac{1}{q_0}.$$

We now consider the group $G = \mathbb{Z}^\ell/(\boldsymbol{\Pi''})$ and define the set

$$U = \{\boldsymbol{a} \neq \boldsymbol{0} : \gcd(a_1, \ldots, a_\tau, d_r) \neq 1\},$$

where d_r is the larger invariant factor of G, *i.e.* is such that $G \simeq \mathbb{Z}/d_1\mathbb{Z} \times \cdots \times \mathbb{Z}/d_r\mathbb{Z}$ with $d_i \mid d_{i+1}$, and $d_i \neq d_{i+1}$. We denote by $U^{\complement}$ the complement of U. From Lemma 5 we have that $\left|\ker \phi_{\boldsymbol{a},\boldsymbol{\Pi''}}\right| = |\det(\boldsymbol{\Pi''})|^{\tau-1}$ when $\boldsymbol{a} \in U^{\complement}$. Therefore,

$$\left|\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] - \frac{1}{q_0\,|\det(\boldsymbol{\Pi''})|}\right| \leqslant \begin{cases} 0 & \text{if } \boldsymbol{b}-\boldsymbol{b'} \in U^{\complement}, \\ \frac{1}{q_0} & \text{if } \boldsymbol{b}-\boldsymbol{b'} \in U. \end{cases}$$

Therefore it suffices to bound the number of $(\boldsymbol{b}, \boldsymbol{b'})$ such that $\boldsymbol{b}-\boldsymbol{b'} \in U$. For any $\boldsymbol{b} \in X$, define

$$U_{\boldsymbol{b}} = \{\boldsymbol{b'} \in X \setminus \{\boldsymbol{b}\} : \boldsymbol{b}-\boldsymbol{b'} \in U\}.$$

We write $d_r = \beta_1 \cdots \beta_m$ with β_j prime for $j \in \{1, \ldots, m\}$, where the β_j's are not supposed distinct. Thus $\boldsymbol{b'} \in U_{\boldsymbol{b}}$ if and only if there exists $j \in \{1, \ldots, m\}$ such that $b_i - b'_i \in \beta_j\mathbb{Z}$ for all $i = 1, \ldots, \tau$. Now, since we have $|(\beta_j\mathbb{Z} \cap (-2^\alpha, 2^\alpha))| \leqslant 2^{\alpha+1}/\beta_j$ and $\beta_j \geqslant 2$, as well as $d_r \leqslant |\det(\boldsymbol{\Pi''})| < 2^{\ell\cdot(\rho'+1)}$ which implies $m \leqslant \ell \cdot (\rho'+1)$, we obtain:

$$|U_{\boldsymbol{b}}| \leqslant \sum_{j=1}^{m} \left(\frac{2^{\alpha+1}}{\beta_j}\right)^{\tau} \leqslant \sum_{j=1}^{\ell\cdot(\rho'+1)} \left(\frac{2^{\alpha+1}}{2}\right)^{\tau} = \ell \cdot (\rho'+1) \cdot 2^{\alpha\tau}.$$

This gives

$$\begin{aligned} \delta &\leqslant \frac{|Y|}{|X|^2} \sum_{\boldsymbol{b}-\boldsymbol{b'} \in U} \frac{1}{q_0} = \frac{|\det(\boldsymbol{\Pi'})|}{|X|^2} \sum_{\boldsymbol{b} \in X} \sum_{\boldsymbol{b'} \in U_{\boldsymbol{b}}} \frac{1}{q_0} \\ &\leqslant \frac{q_0\,|\det(\boldsymbol{\Pi''})|}{q_0 \cdot |X|^2} \sum_{\boldsymbol{b} \in X} |U_{\boldsymbol{b}}| \leqslant \frac{|\det(\boldsymbol{\Pi''})|}{|X|} \cdot \ell \cdot (\rho'+1) \cdot 2^{\alpha\tau} \\ &\leqslant \frac{2^{\ell\cdot(\rho'+1)}}{2^{(\alpha+1)\cdot\tau}} \cdot \ell \cdot (\rho'+1) \cdot 2^{\alpha\tau} \leqslant \ell \cdot \rho' \cdot 2^{\ell\cdot(\rho'+2)-\tau} \end{aligned}$$

This shows that the hash function family $\mathcal{H}$ is ε-pairwise independent, with $\varepsilon := \ell \cdot \rho' \cdot 2^{\ell\cdot(\rho'+2)-\tau}$.

Finally, applying Lemma 10 we obtain that the distribution of $(h, h(\boldsymbol{b}))$ for $h \leftarrow \mathcal{H}$, $\boldsymbol{b} \leftarrow (-2^\alpha, 2^\alpha)^\tau$ is ε_2-uniform over $\mathcal{H} \times Y$, where:

$$\varepsilon_2 := \frac{1}{2}\sqrt{\frac{|Y|}{|X|} + \varepsilon} \leqslant \sqrt{2^{\gamma-\alpha\cdot\tau} + \ell \cdot \rho' \cdot 2^{\ell\cdot(\rho'+2)-\tau}}$$

which proves Lemma 9.

B.7 Proof of Claim 3

We must show that that the distribution

$$\mathcal{D}_1 = \left\{\boldsymbol{u} + \boldsymbol{\Pi} \cdot \boldsymbol{b'} \bmod \boldsymbol{x_0} : \boldsymbol{u} \leftarrow \mathbb{Z}^{\ell+1}/(\boldsymbol{\Pi'}),\ \boldsymbol{b'} \leftarrow (\mathbb{Z} \cap (-2^{\alpha'}, 2^{\alpha'}))^\ell\right\}$$

is statistically close to the distribution:

$$\mathcal{D}_2 = \left\{\boldsymbol{v} + \boldsymbol{\Pi} \cdot \boldsymbol{b'} \bmod \boldsymbol{x_0} : \boldsymbol{v} \leftarrow \mathbb{Z}_{q_0} \times (\mathbb{Z} \cap [0, 2^{\rho'}))^\ell,\ \boldsymbol{b'} \leftarrow (\mathbb{Z} \cap (-2^{\alpha'}, 2^{\alpha'}))^\ell\right\}.$$

We consider the intermediate distribution:

$$\mathcal{D}'_2 = \left\{(\boldsymbol{v} \bmod \boldsymbol{\Pi'}) + \boldsymbol{\Pi} \cdot \boldsymbol{b'} \bmod \boldsymbol{x_0} : \boldsymbol{v} \leftarrow \mathbb{Z}_{q_0} \times (\mathbb{Z} \cap [0, 2^{\rho'}))^\ell,\ \boldsymbol{b'} \leftarrow (\mathbb{Z} \cap (-2^{\alpha'}, 2^{\alpha'}))^\ell\right\}.$$

Applying the same technique as in the proof of Claim 1, the statistical distance between $\mathcal{D}_2$ and $\mathcal{D}'_2$ is at most $\ell \cdot 2^{3-\alpha'}$. Moreover applying the same technique as in the proof of Lemma 8 the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}'_2$ is at most $\ell^2 \cdot 2^{\rho-\rho'+2}$. Summing the two statistical distances proves Claim 3.

C Proof of Lemma 3

We first prove the following lemma.

Lemma 11. *The ℓ-decisional-Approximate-GCD problem is hard if the* 1*-decisional-Approximate-GCD problem is hard.*

Proof. Given an algorithm $\mathcal{A}_\ell$ for solving the ℓ-decisional-Approximate-GCD problem, we construct an algorithm $\mathcal{A}_1$ for solving the 1-decisional-Approximate-GCD problem. Our proof is based on generating $\ell - 1$ of the primes p_i's ourselves and putting the prime p from the 1-decisional instance at a random position among the p_i's; we then show how to simulate the oracle for the ℓ-decisional instance using the oracle from the 1-decisional instance; eventually the challenge for the ℓ-decisional instance is simulated from the 1-decisional challenge using an hybrid argument.

We receive as input an integer $y_0 = p \cdot q_0$ where p is a random η-bit integer and $q_0 \leftarrow [0, 2^\gamma/p)$ is 2^{λ^2}-rough. We select a random integer $i_0 \in [0, \ell - 1]$ and we generate $\ell - 1$ random η-bit primes p_i for $i \in [0, \ell - 1] \setminus \{i_0\}$. We implicitly let $p_{i_0} = p$, where p is unknown. We let:

$$x_0 := y_0 \cdot \prod_{\substack{i=0 \\ i \neq i_0}}^{\ell-1} p_i = q_0 \cdot \prod_{i=0}^{\ell-1} p_i$$

and we send x_0 to $\mathcal{A}_\ell$.

Our simulation of the ℓ-oracle from the 1-oracle works as follows: given an integer y such that $y = 2r + v \bmod p$ we can obtain by CRT an integer z such that $z = 2r + v \bmod p$ and $z = 2r_i + v_i \bmod p_i$ for all $i \neq i_0$, without knowing p but using $y_0 = q_0 \cdot p$. Formally we have the following equality:

$$\mathsf{CRT}_{q_0,(p_i)}(q, u_0, \ldots, u_{\ell-1}) = \mathsf{CRT}_{q_0 \cdot p_{i_0},(p_i)_{i \neq i_0}} \left(\mathsf{CRT}_{q_0,p_{i_0}}(q, u_{i_0}), u_0, \ldots, u_{i_0-1}, u_{i_0+1}, \ldots u_{\ell-1} \right) \tag{20}$$

Therefore given as input a query $\boldsymbol{v} = (v_i)$ to the ℓ-oracle $\mathcal{O}_{q_0,(p_i)}$ we first query the 1-oracle $\mathcal{O}_{q_0,p}$ with v_{i_0} to obtain:

$$y = \mathsf{CRT}_{q_0,p}(q, 2r + v_{i_0})$$

Then we generate random $r_i \in \mathbb{Z} \cap (-2^\rho, 2^\rho)$ for all $i \neq i_0$ and return the following value:

$$z := \mathsf{CRT}_{y_0,(p_i)_{i \neq i_0}} (y, 2r_0 + v_0, \ldots, 2r_{i_0-1} + v_{i_0-1}, 2r_{i_0+1} + v_{i_0+1}, \ldots, 2r_{\ell-1} + v_{\ell-1})$$

From equality (20), we have:

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

which shows that our simulation of the oracle $\mathcal{O}_{q_0,(p_i)}$ is perfect.

It remains to show how we generate the challenge z for the ℓ-instance. We receive as input a challenge $y = \mathsf{CRT}_{q_0,p}(q, 2r + v_b)$ from the 1-instance, where $b \leftarrow \{0, 1\}$, $v_0 = 0$ and $v_1 \leftarrow \{0, 1\}$, and we must output a guess b' of b. From y we construct a hybrid challenge z as previously

$$z = \mathsf{CRT}_{y_0,(p_i)_{i \neq i_0}} (y, 2r_0 + v_0, \ldots, 2r_{i_0-1} + v_{i_0-1}, 2r_{i_0+1} + v_{i_0+1}, \ldots, 2r_{\ell-1} + v_{\ell-1})$$

where $v_i = 0$ for all $0 \leqslant i < i_0$ and $v_i \leftarrow \{0, 1\}$ for all $i_0 < i \leqslant \ell - 1$. Therefore z is an hybrid challenge containing the following bits w_i:

$$z = \mathsf{CRT}_{q_0,(p_i)}(q, 2r_0 + w_0, \ldots, 2r_{\ell-1} + w_{\ell-1})$$

where:

$$w_0 \leftarrow 0, \ldots, \ w_{i_0-1} \leftarrow 0, \ w_{i_0} \leftarrow \begin{cases} 0 & \text{if } b = 0 \\ \{0, 1\} & \text{if } b = 1 \end{cases}, \ w_{i_0+1} \leftarrow \{0, 1\}, \ldots, \ w_{\ell-1} \leftarrow \{0, 1\}$$

We send z to the $\mathcal{A}_\ell$ algorithm; eventually we output the bit b' returned by $\mathcal{A}_\ell$.

To analyze the success probability of our algorithm $\mathcal{A}_1$ we use the following hybrid argument. We note that the distribution of z for $i_0 = j$ and $b = 0$ is the same as the distribution of z for $i_0 = j + 1$ and $b = 1$, for all $0 \leqslant j < \ell - 1$. Moreover $i_0 = 0$ and $b = 1$ corresponds to the ℓ-oracle being called with $\boldsymbol{v} \leftarrow \{0,1\}^\ell$, while $i_0 = \ell - 1$ and $b = 0$ corresponds to $\boldsymbol{v} = 0$. Since i_0 is generated at random in $[0, \ell - 1]$ the distinguishing probability of our algorithm $\mathcal{A}_1$ is given by:

$$\begin{aligned}\Pr[b'=0|b=0] - \Pr[b'=0|b=1] &= \frac{1}{\ell} \cdot \sum_{j=0}^{\ell-1} \Pr[b'=0|i_0 = j \wedge b = 0] - \Pr[b'=0|i_0 = j \wedge b = 1] \\ &= \frac{1}{\ell} \cdot \Big(\Pr[b'=0|\boldsymbol{v} \leftarrow \{0,1\}^\ell] - \Pr[b'=0|\boldsymbol{v}=0]\Big) \\ &= \frac{\varepsilon}{\ell}\end{aligned}$$

where ε is the distinguishing probability of $\mathcal{A}_\ell$ on the ℓ-instance. Therefore the advantage of our algorithm $\mathcal{A}_1$ is non-neglible if the advantage of $\mathcal{A}_\ell$ is non-neligible. This terminates the proof of Lemma 11. □

It remains to show the following lemma.

Lemma 12. *The* 1*-decisional-Approximate-GCD problem is hard if the error-free Approximate-GCD problem is hard.*

Proof (sketch). The proof is essentially the same as the proof of semantic security of the original DGHV scheme in [DGHV10]. Namely from an algorithm $\mathcal{A}_1$ solving the 1-decisional-Approximate-GCD problem one can build a similar "high-accuracy LSB predictor" for given z computing $[z \bmod p]_2$. Then such "high-accuracy LSB predictor" is used to obtain a Binary GCD algorithm for integers of the form $z = q \cdot p + r$ for small r, which eventually enables to recover p. □

D Complete Description of the Batch DGHV Scheme with Compressed Public Keys

In this section, we provide a complete description of our batch FHE scheme with the ciphertext compression technique of [CNT12]. Note that as in [CNT12], the ciphertext compression technique is applied to both the public key elements x_i's, x'_i's and Π_i's of the somewhat homomorphic scheme, and to the encryption σ_i's of the secret key bits. The ciphertext compression technique enables to compress a ciphertext from $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ bits down to $\ell \cdot \eta + \lambda = \ell \cdot \tilde{\mathcal{O}}(\lambda^2)$ bits.

D.1 Description

$\mathsf{BDGHV.KeyGen}(1^\lambda)$**.** Generate a collection of ℓ primes $p_0, \ldots, p_{\ell-1}$ of size η bits, and denote π their product. 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.
Initialize a pseudo-random generator f_1 with a random seed se_1. Use $f_1(\mathsf{se}_1)$ to generate a set of integers $\chi_i \in [0, x_0)$ for $i \in [1, \tau]$, $\chi'_i \in [0, x_0)$ for $i \in [0, \ell - 1]$ and $\chi_i^\Pi \in [0, x_0)$ for $i \in [0, \ell - 1]$. Define γ-bit integers as follows:

1. the integers x_i's ($1 \leqslant i \leqslant \tau$) such that $x_i = \chi_i - \Delta_i$ with

$$\Delta_i = [\chi_i]_\pi + \xi_i \cdot \pi - \mathsf{CRT}_{p_0,\ldots,p_{\ell-1}}(2r_{i,0}, \ldots, 2r_{i,\ell-1})$$

where $r_{i,j} \leftarrow \mathbb{Z} \cap (-2^{\rho'-1}, 2^{\rho'-1})$ and $\xi_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda + \log_2(\ell) + \ell \cdot \eta}/\pi)$;

2. the integers x'_i's ($0 \leqslant i \leqslant \ell - 1$) such that $x'_i = \chi'_i - \Delta'_i$ with
$$\Delta'_i = [\chi'_i]_\pi + \xi'_i \cdot \pi - \mathsf{CRT}_{p_0,\ldots,p_{\ell-1}}(2r'_{i,0} + \delta_{i,0}, \ldots, 2r'_{i,\ell-1} + \delta_{i,\ell-1})$$
where $r'_{i,j} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ and $\xi'_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda+\log_2(\ell)+\ell\cdot\eta}/\pi)$;
3. the integers Π_i's ($0 \leqslant i \leqslant \ell - 1$) such that $\Pi_i = \chi^\Pi_i - \Delta^\Pi_i$ with
$$\Delta^\Pi_i = [\chi^\Pi_i]_\pi + \xi^\Pi_i \cdot \pi - \mathsf{CRT}_{p_0,\ldots,p_{\ell-1}}(2\varpi_{i,0} + \delta_{i,0}2^{\rho'+1}, \ldots, 2\varpi_{i,\ell-1} + \delta_{i,\ell-1}2^{\rho'+1})$$
where $\varpi_{i,j} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ and $\xi^\Pi_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda+\log_2(\ell)+\ell\cdot\eta}/\pi)$.

Additionally, generate at random Θ-bit vectors $\boldsymbol{s}_j = (s_{j,0}, \ldots, s_{j,\Theta-1})$ for $0 \leqslant j < \ell$, each split in θ boxes of size $B = \Theta/\theta$ each, with exactly one non-zero bit in each box, and such that $s_{j,j} = 1$ for each $j = 0, \ldots, \ell - 1$. Initialize a pseudo-random generator f_2 with a random seed se_2, and use $f_2(\mathsf{se}_2)$ to generate integers $u_i \in [0, 2^{\kappa+1})$ for $i \in \mathbb{Z} \cap [0, \Theta - 1]$. Then set $u_0, \ldots, u_{\ell-1}$ such that
$$x_{p_j} = \sum_{i=0}^{\Theta-1} s_{j,i} \cdot u_i \bmod 2^{\kappa+1},$$
where $x_{p_j} = \lfloor 2^\kappa/p_j \rceil$ for each $j = 0, \ldots, \ell - 1$.

Initialize a pseudo-random generator f_3 with a random seed se_3. Use $f_3(\mathsf{se}_3)$ to generate a set of integers $\chi^\sigma_i \in [0, x_0)$ for $i \in [0, \Theta - 1]$.

For $i \in [0, \Theta - 1]$, define the γ-bit integers $\sigma_i = \chi^\sigma_i - \Delta^\sigma_i$ with
$$\Delta^\sigma_i = [\chi^\sigma_i]_\pi + \xi^\sigma_i \cdot \pi - \mathsf{CRT}_{p_0,\ldots,p_{\ell-1}}(2r''_{i,0} + s_{0,i}, \ldots, 2r''_{i,\ell-1} + s_{\ell-1,i}),$$
where $r''_{i,j} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$ and $\xi^\sigma_i \leftarrow \mathbb{Z} \cap [0, 2^{\lambda+\log_2(\ell)+\ell\cdot\eta}/\pi)$.

Output the secret key $\mathsf{sk} = (\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{\ell-1})$ and the public key
$$\mathsf{pk} = \left\langle x_0, \mathsf{se}_1, (\Delta_i)_{1\leqslant i\leqslant\tau}, (\Delta'_i)_{0\leqslant i\leqslant\ell-1}, (\Delta^\Pi_i)_{0\leqslant i\leqslant\ell-1}, \mathsf{se}_2, u_0, \ldots, u_{\ell-1}, \mathsf{se}_3, (\Delta^\sigma_i)_{0\leqslant i<\Theta} \right\rangle.$$

BDGHV.Encrypt($\mathsf{pk}, \boldsymbol{m} \in \{0,1\}^\ell$). Use $f_1(\mathsf{se}_1)$ to recover the integers χ_i's, χ'_i's, and χ^Π_i's. Let $x_i = \chi_i - \Delta_i$ for $1 \leqslant i \leqslant \tau$ and $x'_i = \chi'_i - \Delta'_i$, $\Pi_i = \chi^\Pi_i - \Delta^\Pi_i$ for $0 \leqslant i \leqslant \ell - 1$. Choose random integer vectors $\boldsymbol{b} = (b_i)_{1\leqslant i\leqslant\tau} \in (-2^\alpha, 2^\alpha)^\tau$ and $\boldsymbol{b'} = (b'_i)_{0\leqslant i\leqslant\ell-1} \in (-2^{\alpha'}, 2^{\alpha'})^\ell$ and output the ciphertext:
$$c = \left[\sum_{i=0}^{\ell-1} m_i \cdot x'_i + \sum_{i=1}^{\tau} b_i \cdot x_i + \sum_{i=0}^{\ell-1} b'_i \cdot \Pi_i \right]_{x_0}.$$

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

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

BDGHV.Expand(pk, c). The ciphertext expand procedure takes a ciphertext c and compute the associated expanded ciphertext. To do so, use $f_2(s_2)$ to recover $u_\ell, \ldots, u_{\Theta-1}$, then let $y_i = u_i/2^\kappa$ and 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.Decrypt($\mathsf{sk}, c, \boldsymbol{z}$). Output $\boldsymbol{m} = (m_0, \ldots, m_{\ell-1})$ with
$$m_j \leftarrow \left[\left\lfloor \sum_{i=0}^{\Theta-1} s_{j,i} \cdot z_i \right\rfloor \right]_2 \oplus (c \bmod 2).$$

BDGHV.Recrypt($\mathsf{pk}, c, \boldsymbol{z}$). 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}.

D.2 Semantic Security

We prove the semantic security of the previous scheme. Note that the random oracle model is only necessary when using compressed public keys as in [CNT12]; the semantic security of our batch FHE scheme from Section 3 does not require random oracles.

Theorem 3. *The previous encryption scheme is semantically secure under the EF-ℓ-dec-AGCD assumption, in the random oracle model.*

Proof (sketch). The proof is almost the same as in Section 3.3. Following the same strategy as in [CNT12], the random oracle can be programmed so that the distribution of the public key is statistically close to that of the batch scheme without compressed public keys. □

E Description of our Homomorphic AES Implementations

In this section we describe two implementations of homomorphic evaluation of an AES circuit (HAES), using our BDGHV scheme with ℓ plaintext bits embedded in each ciphertext.

E.1 Byte-Wise Bitslicing

Throughout this subsection, we denote $k = \lfloor \ell/16 \rfloor$ the number of AES-128 encryptions one will be able to perform in parallel. We define a representation called *byte-wise bitslicing* in which the HAES state will be composed of 8 ciphertexts, each ciphertext containing one and exactly one bit of each byte of the AES state. This construction is similar to general-purpose bitslicing [Bih97,KS09].

State. We recall that the AES state is a matrix of 4×4 bytes. It can be viewed as a 16-byte vector when reading the bytes by column.

The state in our representation is composed of 8 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 state of the k-th AES (see Figure 1). Thus $\boldsymbol{m}_0$ represents the LSBs of the bytes of the AES states for the k AES-plaintexts, and $\boldsymbol{m}_7$ the MSBs.

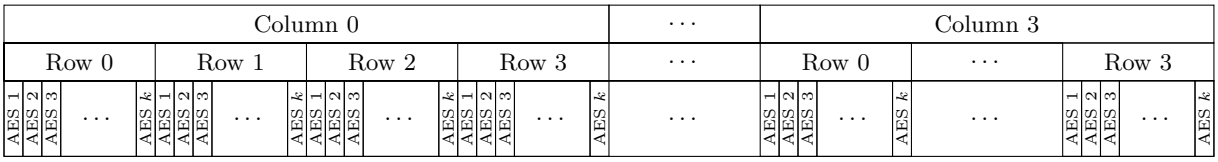

Fig. 1. Bit ordering in $\boldsymbol{m}_i$ in the byte-wise bitslicing representation

AddRoundKey. From the 128-bit AES key, an expanded key is created with $(10+1)$ rounds subkeys (namely, one subkey at the beginning of the encryption, and one per round). Each round subkey is XORed at one point with the AES state. Thus, each bit of the subkey is XORed with the corresponding bit of the state.

In this variant, we construct the round subkeys with the same structure as the HAES state (*i.e.* 8 ciphertexts); notice that we repeat each bit of the round subkey k times. Therefore, the AddRoundKey stage only consists in adding the corresponding ciphertexts with BDGHV.Add as the underlying operation is a XOR on the plaintext bits. This operation consists of 8 BDGHV.Add operations.

SubBytes. The SubBytes stage consists for each byte $\boldsymbol{b}$ to apply the transformation $\boldsymbol{b} \mapsto M\boldsymbol{b}^{254} + \boldsymbol{c}$, where the power function is performed over $\mathrm{GF}(2^8)$,

$$M = \begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{c} = \mathsf{0x63}\ .$$

Notice that the multiplication by M is viewed over $\mathrm{GF}(2)^8 \simeq \mathrm{GF}(2^8)$, viewing $\boldsymbol{b}$ as its bit-representation vector.

A possible way to implement this stage could be to implement the multiplication of two FHE ciphertexts so that each byte of the underlying state is multiplied over $\mathrm{GF}(2^8)$ by the corresponding bit. Each multiplication would consumes one level of noise, *i.e.* after each multiplication 8 recryptions would be needed to recover a state with a "small" noise (*i.e.* small enough so that the eight ciphertexts can be multiplied once without prior recryption). The minimal solution to compute $\boldsymbol{b}^{254}$ from $\boldsymbol{b}$ costs 4 multiplications and several squarings [RP10] (but here the squarings can be obtained easily with several BDGHV.Add only), giving a final cost of 32 BDGHV.Recrypt. However, as seen in Section 4.4, the cost of a recryption is really huge compared to the other operations.

To limit the number of recryptions, we used the 115 gates circuit of Boyar and Peralta [BP10] to compute the Sbox. In this circuit, we also have 32 multiplications, thus one could think that 32 recryptions are necessary. However, we can reduce the number of recryptions to 17 by a careful management of the noise. Recryptions are then performed on temporary variables t_{21}, t_{22}, t_{23}, t_{24}, t_{26}, t_{29}, t_{33}, t_{36}, t_{40} and on the eight outputs; this enables to transform an homomorphic AES state with "small" noise into an homomorphic AES state with "small" noise after the SubBytes operation. If one does not care about the output noises, only 9 recryptions are performed. Therefore, this stage costs 32 BDGHV.Mult, 17 BDGHV.Recrypt and 83 BDGHV.Add. Note that under our representation the SBOX circuit is evaluated in parallel over the 16 bytes of an AES state, and also on the k AES blocks.

ShiftRows. The ShiftRows stage consists in performing the permutation ζ_{SR} on the bytes of the AES state, where the Cauchy's two-line notation of the permutation is

$$\zeta_{\mathrm{SR}} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 13 & 10 & 7 & 4 & 1 & 14 & 11 & 8 & 5 & 2 & 15 & 12 & 9 & 6 & 3 \end{pmatrix}.$$

Since we "sliced" the bytes of the AES state in our representation, we will need to apply a similar permutation on each ciphertext of the state. Since we are performing k AES blocks in parallel, we need to consider the permutation ζ defined by

$$\zeta(I \times k + K) = \zeta_{\mathrm{SR}}(I) \times k + K, \qquad 0 \leqslant K \leqslant k-1, 0 \leqslant I \leqslant 15.$$

Next, we need to permute each of the c_i's of the HAES state by ζ to perform the ShiftRows stage. As mentioned in Section 4.3, rotating the slots is "for free" when performed during a recryption. Now, we perform a recryption on each element of the HAES state at the end of the SubBytes stage. Thus, instead of using the regular σ_i's, we use the σ_i^{ζ}'s and the ShiftRows stage will be obtained at the end of the SubBytes stage at *no additional cost.*

MixColumns. The MixColumns stage consists in in multiplying each column of the AES state (thus a vector of four bytes) by the matrix M' over $\mathrm{GF}(2^8)$, where

$$M' = \begin{pmatrix} 0\text{x}02 & 0\text{x}03 & 0\text{x}01 & 0\text{x}01 \\ 0\text{x}01 & 0\text{x}02 & 0\text{x}03 & 0\text{x}01 \\ 0\text{x}01 & 0\text{x}01 & 0\text{x}02 & 0\text{x}03 \\ 0\text{x}03 & 0\text{x}01 & 0\text{x}01 & 0\text{x}02 \end{pmatrix} .$$

We can view this operation over the whole state as

$$\begin{pmatrix} s'_0 & s'_4 & s'_8 & s'_{12} \\ s'_1 & s'_5 & s'_9 & s'_{13} \\ s'_2 & s'_6 & s'_{10} & s'_{14} \\ s'_3 & s'_7 & s'_{11} & s'_{15} \end{pmatrix} = 0\text{x}02 \times \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix} \oplus 0\text{x}03 \times \begin{pmatrix} s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \\ s_0 & s_4 & s_8 & s_{12} \end{pmatrix} \oplus \begin{pmatrix} s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \\ s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \end{pmatrix} \oplus \begin{pmatrix} s_3 & s_7 & s_{11} & s_{15} \\ s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \end{pmatrix} .$$

Thus in our implementation, we define three copies of the HAES state (*i.e.* 24 additional ciphertexts) and we rotate them according to the permutations ζ_1, ζ_2 or ζ_3, with

$$\zeta_i(I \times k + K) = \zeta_{\mathrm{MC}_i}(I) \times k + K, \qquad 0 \leqslant K \leqslant k-1, 0 \leqslant I \leqslant 15,$$

the permutation ζ_{MC_i} being defined as

$$\zeta_{\mathrm{MC}_i}(I) = \lfloor I/4 \rfloor \cdot 4 + (I + i \bmod 4), \qquad 0 \leqslant I \leqslant 15 .$$

This costs $3 \times 8 = 24$ BDGHV.Recrypt. Next, we need to multiply by 0x02 the current HAES state and by 0x03 the first copy.

Algorithm 1 Multiplication by 0x02 in $\mathrm{GF}(2^8)$

Input: A element $\boldsymbol{b} = \sum_{i=0}^{7} b_i x^i$ of $\mathrm{GF}(2)[x]$
Output: The product $\boldsymbol{b} \cdot 0\text{x}02$ over $\mathrm{GF}(2)[x]/(x^8 + x^4 + x^3 + x + 1)$
1: Compute $b'_0 = b_7$
2: Compute $b'_1 = b_0 \oplus b_7$
3: Compute $b'_2 = b_1$
4: Compute $b'_3 = b_2 \oplus b_7$
5: Compute $b'_4 = b_3 \oplus b_7$
6: Compute $b'_5 = b_4$
7: Compute $b'_6 = b_5$
8: Compute $b'_7 = b_6$
9: **return** $\sum_{i=0}^{7} b'_i x^i$

Algorithm 2 Multiplication by 0x03 in $\mathrm{GF}(2^8)$

Input: A element $\boldsymbol{b} = \sum_{i=0}^{7} b_i x^i$ of $\mathrm{GF}(2)[x]$
Output: The product $\boldsymbol{b} \cdot 0\text{x}03$ over $\mathrm{GF}(2)[x]/(x^8 + x^4 + x^3 + x + 1)$
1: **return** $\boldsymbol{b} \cdot 0\text{x}02 + \boldsymbol{b}$

Multiplications by such values over $\mathrm{GF}(2^8)$ are easy to obtain (see Algorithms 1 and 2). When applying these algorithms on the HAES state $(c_0, \ldots, c_7)$ instead of $(b_0, \ldots, b_7)$, the multiplication by 0x02 is performed in parallel over all the bytes of an underlying AES state, but also on the k AES states. Therefore, $3 + 11 = 14$ BDGHV.Add operations are performed during this step.

Finally, we need to add the four copies (possibly rotated or multiplied) of the state to get the final HAES state, and this is performed in $3 \times 8 = 24$ BDGHV.Add.

Final Cost. The AES encryption process consists of 11 AddRoundKey stages, 10 SubBytes and ShiftRows stages and 9 MixColumns stages. Therefore, the final cost is 1260 BDGHV.Add, 386 BDGHV.Recrypt and 320 BDGHV.Mult. However, a fine management of the noise allows us to reduce the number of BDGHV.Recrypt to 377.

E.2 State-Wise Bitslicing

Throughout this subsection, we will be able to perform ℓ AES-128 encryptions in parallel. We define a representation called *state-wise bitslicing* in which the HAES state will be composed of 128 ciphertexts, each ciphertext containing one and exactly one bit of the AES state.

State. The state in our representation 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 element of the state of the k-th AES. In other words, the 128 bits of the State are represented in 128 different ciphertexts, and since ℓ slots are available, one can put ℓ different AES states in the 128 ciphertexts.

AddRoundKey. The AddRoundKey stage simply consists of BDGHV.Add operations, one per ciphertext in the state. Therefore, this stage costs 128 BDGHV.Add.

SubBytes. As above, we use the Boyar and Peralda circuit [BP10] to perform this operation. However, our representation is less adapted to this stage than previously, since even though we process the k AES blocks in parallel, we need to apply this SubBytes stage on each group of 8 ciphertexts corresponding to one of the 16 bytes of the AES state. Therefore, we need to apply the same circuit 16 times with different inputs. Therefore, this stage costs $16 \times 83 = 1328$ BDGHV.Add, $16 \times 17 = 272$ BDGHV.Recrypt and $16 \times 32 = 512$ BDGHV.Mult.

ShiftRows. The permutation of the ShiftRows stage is no longer applied on the plaintext bits in each ciphertext of the State, but on the indices of the ciphertext of the HAES state. This stage is a relabelling of the indices of the ciphertexts of the HAES state. Therefore, this operation does not cost any homomorphic operation.

MixColumns. As above, the new HAES state can be written as a sum of four (possibly rotated or multiplied) states. But as in the ShiftRows stage, the permutations are no longer applied on the ciphertexts to rotate the plaintext bits, but on the indices of the ciphertexts of the HAES state. Therefore, copying and permuting the states is essentially free. However, we still need to multiply by 0x02 and 0x03 the current HAES state and its first permuted copy respectively. Algorithms 1 and 2 are still valid, and allows us to perform such multiplications in $(3 + 11) \times 16 = 224$ BDGHV.Add. Finally, we need to add the 4×128 ciphertexts, and this costs $3 \times 128 = 384$ BDGHV.Add.

Final Cost. The AES encryption process consists of 11 AddRoundKey stages and 10 SubBytes. Therefore, the final cost is 14688 BDGHV.Add, 2720 BDGHV.Recrypt and 5120 BDGHV.Mult. However, a fine management of the noise allows us to reduce the number of BDGHV.Recrypt to 2448 (namely, we do not need to bootstrap *at all* in SubBytes during the first round).

Even though this representation seems far more costly in number of homomorphic operations, notice that in the state-wise bitslicing representation, we can process 16 times more AES plaintexts in parallel than in the byte-wise bitslicing representation. Therefore, for the same number of AES plaintexts, we save 60% of BDGHV.Recrypt, which gives better relatives times.