**Article:**
Dyer, J, Dyer, M and Xu, J orcid.org/0000-0002-4598-167X (2019) Practical homomorphic encryption over the integers for secure computation in the cloud. International Journal of Information Security, 18 (5). pp. 549-579. ISSN 1615-5262

https://doi.org/10.1007/s10207-019-00427-0

**Noname manuscript No.**
(will be inserted by the editor)

# Practical Homomorphic Encryption Over the Integers for Secure Computation in the Cloud

**James Dyer · Martin Dyer · Jie Xu**

the date of receipt and acceptance should be inserted later

**Abstract** We present novel homomorphic encryption schemes for integer arithmetic, intended primarily for use in secure single-party computation in the cloud. These schemes are capable of securely computing arbitrary degree polynomials homomorphically. In practice, ciphertext size and running times limit the polynomial degree, but this appears sufficient for most practical applications. We present four schemes, with increasing levels of security, but increasing computational overhead. Two of the schemes provide strong security for high-entropy data. The remaining two schemes provide strong security regardless of this assumption. These four algorithms form the first two levels of a hierarchy of schemes and we also present the general cases of each scheme. We further elaborate how a fully homomorphic system can be constructed from one of our general cases. In addition, we present a variant based upon Chinese Remainder Theorem (CRT) secret sharing. We detail extensive evaluation of the first four algorithms of our hierarchy by computing low-degree polynomials. The timings of these computations are extremely favourable by comparison with even the best of existing methods, and dramatically out-perform many well-publicised schemes. The results clearly demonstrate the practical applicability of our schemes.

A preliminary version of this paper [39] was presented at IMACC 2017.

J. Dyer
De Montfort University, UK

M. Dyer · J. Xu
School of Computing, University of Leeds, UK.

## 1 Introduction

With services like Amazon's Elastic MapReduce [4] and Microsoft's HDInsight [59] offering large-scale distributed cloud computing environments, computation in the cloud is becoming increasingly more available. Such services allow for computation on large volumes of data to be performed without the large investment in local computing resources. However, where the data that is processed is sensitive, such as financial or medical data, then uploading such data in its raw form to such a third-party service becomes problematic.

To take advantage of these cloud services, we require a means to process the data securely on such a platform. We designate such a computation, *secure computation in the cloud* (SCC). SCC should not expose input or output data to any other party, including the cloud service provider. Furthermore, the details of the computation should not allow any other party to deduce its inputs and outputs. Cryptography seems the natural approach to this problem.

However, it should be noted that van Dijk and Juels [36] show that cryptography alone cannot realise secure *multi-party* computation in the cloud, where the parties jointly compute a function over their inputs while keeping their own inputs private. Since our approach is via homomorphic encryption, we will restrict our attention to what we will call *secure single-party computation in the cloud* (SSCC).

*Homomorphic encryption* (HE) appears to offer a solution to the SSCC problem. First defined by Rivest et al. [70] in 1978, HE allows a function to be computed on encrypted inputs without ever decrypting the inputs. Suppose we wish to compute the function $f$ on inputs

$x_1, x_2, \ldots, x_n$, then, under HE,

$$\mathsf{Dec}(f'(x'_1, x'_2, \ldots, x'_n)) = f(x_1, x_2, \ldots, x_n),$$

where $x'_1, \ldots, x'_n$ are the encryptions of $x_1, \ldots, x_n$, $f'$ is the equivalent of $f$ in the ciphertext space, and $\mathsf{Dec}$ is the decryption function. HE clearly satisfies some of the requirements for secure computation in the cloud. A *somewhat HE* scheme (SWHE) is a scheme which is homomorphic for only limited inputs and functions. *Fully HE* (FHE) is a scheme that is homomorphic for all functions and inputs. This was first realised by Gentry in 2009 [44], and appeared to be the ideal HE scheme.

However, despite the clear advantages of FHE, and many significant advances [20,18], it remains largely impractical. Two implementations of recent FHE schemes, HELib [49] and FHEW [37], both perform very poorly in practice, both in their running time and space requirements (see section 2.6).

In this paper, we present four novel SWHE schemes for encryption of integers that are additively and multiplicatively homomorphic. These schemes are capable of computing arbitrary degree polynomials. In section 2, we present our usage scenario, a summary of our results, and a discussion of related work. We present our initial homomorphic scheme in section 3, in two variants, HE1 and HE1N. HE1 (section 3.1) provides strong security for integers distributed with sufficient entropy. This security derives from the assumed hardness of the *partial approximate common divisor problem* (PACDP). HE1N (section 3.2) guarantees strong security for integers not distributed with sufficient entropy or where the distribution is not known, by adding an additional "noise" term. In addition to the hardness assumption, we prove that HE1N is IND-CPA secure [6]. Section 4 describes a further two variants, HE2 and HE2N, which increase the entropy of the plaintext by adding a dimension to the ciphertexts, which are 2-vectors. This further increases the security of these schemes by effectively doubling the entropy. HE2 (section 4.1) deals with integers of sufficient entropy, HE2N (section 4.2) with integers without the required entropy or of unknown distribution. HE2N also satisfies IND-CPA. We describe this in some detail, since it appears to be practically useful, and is the simplest version of our general scheme. In section 5, we generalise HE2 and HE2N from 2-vectors to $k$-vectors, for arbitrary $k$, in the scheme HE$k$, with noisy variant HE$k$N. These schemes may also be practical for small enough $k$. In section 6, we detail a variant of our HE2N scheme that employs Chinese Remainder Theorem secret sharing to allow one to process the computation by dividing into subcomputations on smaller moduli. In section 7, we show how to construct a fully homomorphic system from our HE$k$ scheme.

We have performed extensive experimental evaluation of the four schemes presented in this paper. We report on this in section 8. Our results are extremely favourable when compared with other methods. In some cases, our algorithms outperform the running times of directly comparable schemes by a factor of up to 1000, and considerably more than that for fully homomorphic schemes, used in the same context. Finally, in section 9, we conclude the paper.

## 2 Background

### 2.1 Scenario

As introduced above, our work concerns secure single-party computation in the cloud. In our scenario, a secure client wishes to compute a function on a large volume of data. This function could be searching or sorting the data, computing an arithmetic function of numeric data, or any other operation. We consider here the case where the client wishes to perform arithmetic computations on numeric data. This data might be the numeric fields within a record, with non-numeric fields being treated differently.

The client delegates the computation to the cloud. However, while the data is in the cloud, it could be subject to snooping, including by the cloud provider. The client does not wish to expose the input data, or the output of the computation, to possible snooping in the cloud. A snooper here will be a party who may observe the data and the computation in the cloud, but cannot, or does not, change the data or insert spurious data. (In our setting data modification would amount to pointless vandalism.) The snooping may be casual, displaying an uninvited interest, or malicious, intending to use data for the attacker's own purposes.

To obtain the required data privacy, the client's function will be computed homomorphically on an encryption of the data. The client encrypts the source data using a secret key and uploads the encryption to the cloud, with a homomorphic equivalent of the target computation. The cloud environment performs the homomorphic computation on the encrypted data. The result of the homomorphic computation is returned to the client, who decrypts it using the secret key, and obtains the output of the computation.

In this scenario, the source data is never exposed in the cloud, but encryptions of it are. A snooper may observe the computation of the equivalent homomorphic function in the cloud environment. As a result, they may be able to deduce what operations are performed, even though they do not know the inputs. A snooper may also be able to inspect the (encrypted) working

data generated by the cloud computation, and even perform side computations of their own. However, snoopers have no access to the secret key, so cannot make encryptions of their own.

### 2.2 Definitions and Notation

$x \leftarrow_{\$} S$ denotes a value $x$ chosen uniformly at random from the discrete set $S$.

$x \leftarrow_{\mathsf{p}} S$ denotes a prime number $x$ chosen uniformly at random from the discrete set $S$.

$\mathsf{KGen} : \mathcal{S} \to \mathcal{K}$ denotes the key generation function operating on the security parameter space $\mathcal{S}$ and whose range is the secret key space $\mathcal{K}$.

$\mathsf{Enc} : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ denotes the symmetric encryption function operating on the plaintext space $\mathcal{M}$ and the secret key space $\mathcal{K}$, whose range is the ciphertext space $\mathcal{C}$.

$\mathsf{Dec} : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$ denotes the symmetric decryption function operating on the ciphertext space $\mathcal{C}$ and the secret key space $\mathcal{K}$, whose range is the plaintext space $\mathcal{M}$.

$\mathsf{Add} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ denotes the homomorphic addition function whose domain is $\mathcal{C}^2$ and whose range is $\mathcal{C}$.

$\mathsf{Mult} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ denotes the homomorphic multiplication function whose domain is $\mathcal{C}^2$ and whose range is $\mathcal{C}$.

$m$, $m_1$, $m_2$, ... denote plaintext values, and $c$, $c_1$, $c_2$, ... denote ciphertext values.

If $k^* = \binom{k+1}{2}$, $\mathbf{v}_\star = [v_1\ v_2\ \ldots\ v_{k^*}]^T$ denotes a $k^*$-vector which augments the $k$-vector $\mathbf{v} = [v_1\ v_2\ \ldots\ v_k]^T$ by appending elements $v_i = f_i(v_1, \ldots, v_k)$ $(i \in [k+1, k^*])$, for a linear function $f_i$. (All vectors are column vectors throughout.)

$\mathbf{e}_i$ denotes the $i$th unit vector $(i = 1, 2, \ldots)$, with size determined by the context.

$[x, y]$ denotes the integers between $x$ and $y$ inclusive, and $[x, y)$ denotes $[x, y] \setminus \{y\}$.

log denotes $\log_e$ and lg denotes $\log_2$.

If $\lambda$ is a security parameter, "with high probability" will mean with probability $1 - 2^{-\epsilon\lambda}$, for some constant $\epsilon > 0$.

Polynomial time or space will mean polynomial in the security parameter $\lambda$.

### 2.3 Formal Model of Scenario

We have $n$ integer inputs $m_1, m_2, \ldots, m_n$ distributed in $[0, M)$ according to a probability distribution $\mathcal{D}$. If $X$ is a random integer sampled from $\mathcal{D}$, let $\Pr[X = i] = \xi_i$, for $i \in [0, M)$. We will consider three measures of the *entropy* of $X$, measured in bits:

Shannon: $H_1(X) = -\sum_{i=0}^{M-1} \xi_i \lg \xi_i$,
Collision: $H_2(X) = -\lg\left(\sum_{i=0}^{M-1} \xi_i^2\right)$,
Min: $H_\infty(X) = -\lg\left(\max_{i=0}^{M-1} \xi_i\right)$.

It is known that $H_1(X) \geq H_2(X) \geq H_\infty(X)$, with equality if and only if $X$ has the uniform distribution on $[0, M)$, in which case all three are $\lg M$. We will denote $H_\infty(X)$ by $\rho$, so it also follows that $H_1(X), H_2(X) \geq \rho$. We use the term "entropy" without qualification to mean min entropy, $H_\infty(X)$. Note that $H_\infty(X) = \rho \geq \lg M$ implies $\xi_i \leq 2^{-\rho}$, $i \in [0, M)$, and that $M \geq 2^\rho$.

We wish to compute a multivariate polynomial $P$ of degree $d$ on these inputs. A secure client $A$ selects an instance $\mathsf{Enc}(K, \cdot)$ of the encryption algorithm $\mathsf{Enc}$ using the secret parameter set $K$. $A$ encrypts the $n$ inputs by computing $c_i = \mathsf{Enc}(K, m_i)$, for $i \in [1, n]$. $A$ uploads $c_1, c_2, \ldots, c_n$ and $P'$ to the cloud computing environment, where $P'$ is the homomorphic equivalent of $P$ in the ciphertext space. The cloud environment computes $P'(c_1, c_2, \ldots, c_n)$. $A$ retrieves $P'(c_1, c_2, \ldots, c_n)$ from the cloud, and computes

$$P(m_1, m_2, \ldots, m_n) = \mathsf{Dec}(K, P'(c_1, c_2, \ldots, c_n)).$$

A snooper is only able to inspect $c_1, c_2, \ldots, c_n$, the function $P'$, and the computation of $P'(c_1, c_2, \ldots, c_n)$, including subcomputations and working data, and perform side-computations on these.[1] Thus the snooper is *passive* or *honest-but-curious* [45].

### 2.4 Observations from Scenario

First, we can observe that symmetric key encryption is sufficient for the model. Public key encryption is not necessary because there is no key escrow or distribution problem. Additionally, even though the public parameters of our symmetric schemes are exposed to the cloud, they do not provide an encryption oracle.

Note that the $n$ inputs do not necessarily need to be uploaded at once, but $n$ is an upper bound on the total number of inputs. For example, if the polynomial is separable we might compute it in separate stages, and this might be useful in more dynamic situations.

This model is clearly susceptible to certain attacks. We consider ciphertext only, brute force, and cryptanalytic attacks. To avoid cryptanalytic attacks, we must choose the parameters of the system carefully. Here, a brute force attack will mean guessing the plaintext associated with a ciphertext. In our encryption schemes, it will be true that a guess can be verified. Since $\xi_i \leq 2^{-\rho}$ for $i \in [0, M)$, the expected number $\mu$ of guesses before

[1] However, note that our "N" schemes below provide security against more malicious snooping.

making a correct guess satisfies $\mu \geq 2^{\rho}$. Massey [58] gave a corresponding result in terms of the Shannon entropy $H_1(X)$.

Similarly, probability of any correct guess in $2^{\rho/2}$ guesses is at most $2^{-\rho/2}$. This bound holds if we need only guess one of $n$ inputs, $m_1, m_2, \ldots, m_n$, even if these inputs are not independent. Therefore, if $\rho$ is large enough, a brute force attack is infeasible. An example of high entropy data is salaries for a large national or multinational business. Low entropy data might include enumerated types, such as gender.

In our model, known plaintext attack (KPA) is possible only by brute force, and not through being given a sample of plaintext, ciphertext pairs. Chosen plaintext attack (CPA) or chosen ciphertext attack (CCA) do not appear relevant to our model. Since $\mathsf{Enc}(K, \cdot)$ is never exposed in the cloud, there is no realistic analogue of an encryption or decryption oracle, as required by these attacks. In public key encryption, an encryption algorithm is available as part of the system, so CPA should be forestalled, though failure to satisfy IND-CPA [7] does not imply that we can break the system.

Following [6], it is common in studying symmetric key encryption to suppose that, in most practical settings, defence against CPA or CCA is necessary. While IND-CPA and IND-CCA are clearly desirable properties for a cryptosystem, their necessity, in the symmetric key context, seems hard to justify. Both [9] and [14] provide examples intended to support this convention. However, these examples are unconvincing. Nevertheless, we show that the "N" variants of our HE schemes below do satisfy IND-CPA.

We note that observation of the function $P'$, which closely resembles $P$, might leak some information about its inputs. However, we assume that this information is far too weak to threaten the security of the system, as is common in the HE literature. However, if the threat is significant, "garbled circuits" [45] are a possible solution.

Finally, we note that our model of SSCC is very similar to the model of *private single-client computing*, described in [36]. Furthermore, they describe an example practical application, a privacy preserving tax return preparation program, which computes the relevant statistics on government servers without revealing the client's inputs. Another example, cited in [57], is a device which collects health data which is streamed to the cloud. Statistics are computed on the data and reported back to the device. To protect the patient's privacy this data is encrypted by the device and the computations are performed homomorphically. Erkin et al. [41] employ a similar scenario in the description of their privacy-preserving face recognition algorithm.

### 2.5 Our Results

We describe new practical HE schemes for the encryption of integers, to be employed in a SSCC system inspired by the HE scheme CryptDB [65]. CryptDB encrypts integers using the Paillier cryptosystem [63] which is additively homomorphic[2]. Similar systems [76, 77] use ElGamal [40] to support multiplications. The "unpadded" versions of these schemes must be used. These are not secure under CPA [47], reducing the advantage of a public-key system. These schemes do not support both addition and multiplication. Computing the inner product function requires re-encrypting the data once the multiplications have been done, so that the additions can be performed. In a SSCC system, this requires shipping the data back to the initiator for re-encryption, a significant communication overhead. We aim to support both addition and multiplication without this overhead. It should also be noted that a hybrid scheme of Paillier and ElGamal, for a given modulus, will be limited in the degree of polynomials that can be computed. Should a product or sum exceed the modulus then the result cannot be successfully decrypted.

Our scheme is inspired by the SWHE scheme of van Dijk et al. that is used as the basis for a public-key system. As in their system, we add multiples of integers to the plaintext to produce a ciphertext. However, [35] supports only arithmetic mod 2. We generalise their scheme to larger moduli.

We showed above that the input data must have sufficient entropy to negate brute force attacks. If the data lacks sufficient entropy, we will introduce more in two ways. The first adds random "noise" of sufficient entropy to the ciphertext, to "mask" the plaintext. This approach is employed in [35]. In our "N" variants below, we add a random multiple (from 0 to $\kappa$) of a large integer, $\kappa$, to the ciphertext, such that $m_i < \kappa$, for all $i \in [1, N]$. If the entropy of the original data was $\rho$, it becomes $\rho + \lg \kappa$. Therefore, if $\kappa$ is large enough, our data has sufficient entropy. But there is a downside. If the noise term grows too large, the ciphertext cannot be decrypted successfully. So we are restricted to computing polynomials of bounded degree, but this does not appear to be a practical problem.

The other technique will be to increase the dimension of the ciphertext. We represent the ciphertext as a $k$-vector, where each element is a linear function of the plaintext. Addition and multiplication of ciphertexts use linear algebra. The basic case $k = 1$ is described in section 3.1. Then we can increase the en-

[2] Paillier supports computation of linear functions with known coefficients homomorphically by repeated addition

tropy by creating a $k$-vector ciphertext. Then we must guess $k$ plaintexts to break the system. Assuming that the inputs $m_1, m_2, \ldots, m_n$ are chosen independently from $\mathcal{D}$, and the entropy is $\rho$, the entropy of a $k$-tuple $(m_1, m_2, \ldots, m_k)$ is $k\rho$. Thus the $k$-vectors effectively have entropy $k\rho$. If $k$ is chosen large enough, we have sufficient entropy to prevent brute force attack. The assumption of independence among $m_1, m_2, \ldots, m_n$ can be relaxed, to allow some correlation, but we will not discuss the details. On the upside, some cryptanalytic attacks for $k = 1$ do not seem to generalise even to $k = 2$. The downside is that ciphertexts are $k$ times larger, and each homomorphic multiplication requires $\Omega(k^3)$ time and space. For very large $k$, this probably renders the methods impractical. Therefore, we consider the case $k = 2$ in section 4. The general case is considered in section 5.

Our work here supports computing arbitrary degree multivariate polynomials on integer data. However, we expect that for many practical applications, computing low-degree polynomials will suffice. See [57] for a discussion regarding this. In this paper, we present four variants of our scheme. Two provide strong security under the assumption that the input data has high entropy. The other two provide strong security regardless of this assumption. Section 5 generalises these four schemes to dimension $k$ ciphertexts.

### 2.6 Related Work

A comprehensive survey of partial, somewhat, and fully HE schemes is presented in [1]. In this section, we discuss those most related to our own work. Some related work ([65,76,77]) has already been discussed in section 2.5.

Our scheme is inspired by that of van Dijk et al. [35]. In their paper they produce a fully homomorphic scheme over the integers where a simple "somewhat" homomorphic encryption scheme is "bootstrapped" to a fully homomorphic scheme. van Dijk et al. take a simple symmetric scheme where an integer plaintext $m$ is encrypted as $c = m + 2r + pq$, where $p$, the secret key, is an odd $\eta$-bit integer from the interval $[2^{\eta-1}, 2^{\eta})$, and $r$ and $q$ are integers chosen randomly from an interval such that $2r < p/2$. The ciphertext $c$ is decrypted by the calculation $(c \bmod p) \bmod 2$. Our scheme HE1N below (section 3.2) may be regarded as a generalisation of theirs to arbitrary prime moduli.

van Dijk et al. transform their symmetric scheme into a public key scheme. A public key $\langle x_0, x_1, \ldots, x_\tau \rangle$ is constructed where each $x_i$ is a near multiple of $p$ of the form $pq + r'$ where $q$ and $r'$ are random integers chosen from a prescribed interval. To encrypt a message a subset $S$ of $x_i$ from the public key are chosen and the ciphertext is now calculated as $c = m + 2r + 2\sum_{i \in S} x_i \bmod x_0$. The ciphertext is decrypted as previously described. We could extend our HE$k$N schemes here to a public key variant, using a similar device. However, we do not do so, since public key systems appear to have very little application in our model.

van Dijk et al. bootstrap their public key system using Gentry's method [44] to a fully homomorphic scheme. In this case, the bootstrapping is done by homomorphically making a suitable simulation of division by $p$, thus obtaining an encryption of $c \bmod p$ which can be used to continue the computation. Our FHE proposal is based on entirely different principles.

Coron et al. [25,30,31,32] have produced several refinements of the scheme in [35]. In [31], the authors reduce the size of the public key by using a similar but alternative encryption scheme. In this scheme, $p$ is a prime in the specified interval, $x_0$ is an exact multiple of $p$ and the sum term in the ciphertext is quadratic rather than linear. In [32], they apply the Brakerski et al. [18] modulus switching technique to their system from [31]. In [25], the authors apply the Smart and Vercauteren optimisations [74] to their scheme. Finally, in [30], they apply Brakerski's scale-invariant technique [17] to their system. Pisa et al. [64] generalise van Dijk et al.'s scheme from base 2 to base $B$ integers. This scheme is similar to our HE1 scheme. However, our HE1 is a generalisation of van Dijk et al.'s SHE scheme to an arbitrary prime base, rather than a generalisation of the public key scheme. Ramaiah and Kumari [67] produce a variant of van Dijk et al.'s scheme with significantly smaller key sizes. Chen et al. [23] propose a more efficient re-encryption scheme that enhances van Dijk et al.'s scheme. Aggarwal et al. [2] devise a variant without bootstrapping. Nuida and Kurosawa [62] produce a scheme for non binary messages. Most recently, Wang et al. [83] have produce a variant similar to Coron et al.'s original scheme, which reduces the public key size by making the sum term cubic.

Several implementations of SHE and FHE schemes have been produced. Lauter et al. [57] implement the SHE scheme from [20]. However, they give results only for degree two polynomials. Our schemes are capable of computing degree three and four polynomials for practical key and ciphertext sizes. HELib [49] is an implementation of the BGV [18] FHE scheme. HELib-MP [69] is an adaptation of HELib to support multi-precision moduli. At the current time, it only supports basic SHE features. The HEAT (Homomorphic Encryption Applications and Technology) project's HE-API [80] (Homomorphic Encryption Application Programming Interface) has currently integrated HELib and

FV-NFLib [33], an implementation of the Fan and Vercauteren (FV) [42] SHE scheme, under a single API. The authors appear to have made significant improvements in circuit evaluation times, but few details have been made available [15]. Microsoft's SEAL library [56] also implements the FV scheme, albeit, in a modified form. FHEW [38] implements the FHE scheme given in [37]. The performance of these implementations is discussed in section 8.

Erkin et al. [41] exploit the linearly-homomorphic properties of Paillier to compute feature vector matches in their privacy-preserving face recognition algorithm. Our schemes can likewise compute known linear functions, simply by not encrypting the coefficients of the function.

Catalano et al. [22] aim to extend a linearly homomorphic system, such as Paillier [63], to compute multivariate quadratics homomorphically. However, their extension relies on pre-computing a product for each pair of plaintexts and then applying a linear function on the encryption of these products. As such, it does not extend the underlying linear encryption scheme and is not multiplicatively homomorphic. They claim that their system can compute any degree 2 polynomial with at most one multiplication. However, it is not clear how they would compute the polynomial $m_1 \cdot (m_2 + \ldots + m_n)$ without performing $n-1$ offline multiplications. By contrast, our scheme would only require one multiplication. In [21], Catalano et al. extend their approach to cubics.

Zhou and Wornell [85] construct a scheme based on integer vectors, similar, in some respects, to our HE2 (section 4.1) and HE$k$ (section 5) schemes. Bogos et al. [12,13] demonstrate that the system displays some theoretical insecurities. However, the question of whether these are of practical importance is not addressed.

The symmetric MORE scheme [54] uses linear transformations, as do our schemes but in a different way. MORE has been shown [81] to be insecure against KPA, at least as originally proposed. However, whether KPA is relevant in applications of the scheme is unclear.

Recent work on *functional encryption* [46] should also be noted. While these results are of great theoretical interest, the scenario where such schemes might be applied is rather different from our model. Also, the methods of [46] seem too computationally expensive to be of practical interest in the immediate future.

We also note the work of Cheon et al. [26]. They use the Chinese Remainder Theorem (CRT) in an HE system. We make use of the CRT in our scheme HE2NCRT below (section 6). However, our construction differs significantly from theirs.

## 3 Initial Homomorphic Scheme

In this section we present details of our initial SWHE schemes over the integers.

### 3.1 Sufficient Entropy (HE1)

We have $n$ integer inputs $m_1, m_2, \ldots, m_n \in [0, M)$. Negative integers can be handled as in van Dijk et al. [35], by taking residues in $[-(p-1)/2, (p-1)/2)$, rather than $[0, p)$. We wish to compute a polynomial $P$ of degree $d$ in these inputs. The inputs are distributed with entropy $\rho$, where $\rho$ is large enough, as discussed in section 2.3 above. In practical terms, $\rho \geq 32$ will provide sufficient entropy for strong security, since breaking the system would require more than a billion guesses. Our HE scheme is the system (KGen, Pgen, Enc, Dec, Add, Mult).

#### *3.1.1 Key and Parameter Generation*

Let $\lambda$ be a security parameter, measured in bits. Let $p$ and $q$ be randomly chosen large distinct primes such that $p \in [2^{\lambda-1}, 2^{\lambda}]$, and $q \in [2^{\eta-1}, 2^{\eta}]$, where $\eta \approx \lambda^2/\rho - \lambda$. Here $\lambda$ must be large enough to negate direct factorisation of $pq$ (see [55]), and $p$ and $q$ are chosen to negate Coppersmith's attack [29]. We will also require $p > (n+1)^d M^d$ to ensure that $P(m_1, m_2, \ldots, m_n) < p$, so that the result of the computation can be successfully decrypted. Our bounds are worst case, allowing for polynomials which contain all possible monomial terms. For some applications, they will be much larger than required to ensure that $P(m_1, m_2, \ldots, m_n) < p$ and smaller bounds will suffice. Our algorithms KGen (Algorithm 1) and Pgen (Algorithm 2) will randomly select $p$ and $q$ according to these bounds. Then $p$ is the private symmetric key for the system and $pq$ is the modulus for arithmetic performed by Add and Mult. $pq$ is a public parameter of the system. We assume that the entropy $\rho \gg \lg \lambda$, so that a brute force attack cannot be carried out in polynomial time.

**Algorithm 1:** KGen: Key Generation Algorithm

**Input :** $\lambda \in \mathcal{S}$
**Output:** $p \in \mathcal{K}$: secret key
1 $p \leftarrow_{\mathsf{p}} [2^{\lambda-1}, 2^{\lambda}]$
2 **return** $p$

**Algorithm 2:** Pgen: Parameter Generation Algorithm

**Input** : $\lambda \in \mathcal{S}$
**Input** : $\rho \in \mathbb{Z}$: entropy of inputs
**Input** : $p$: secret key
**Output:** modulus $\in \mathbb{Z}$: public modulus
1 $\eta \leftarrow \lambda^2/\rho - \lambda$
2 $q \leftarrow_{\mathsf{p}} [2^{\eta-1}, 2^{\eta}]$
3 modulus $\leftarrow pq$
4 **return** modulus

*3.1.2 Security parameters*

We can easily set the security parameters $\lambda$ and $\eta$ to practical values. To recap, $n$ is the number of inputs, $M$ is an exclusive upper bound on the inputs, $d$ is the degree of the polynomial we wish to calculate. We take $p \approx 2^{\lambda}$ and then $q \approx 2^{\eta}$, where $\eta = \lambda^2/\rho - \lambda$, to guard against the attacks of [28,50].

For HE1, we assume $M \approx 2^{\rho}$, $n \leq \sqrt{M}$. Therefore,

$p > (n+1)^d M^d \approx (nM)^d$ for large $n$.

So, we may take

$p = 2^{\lambda} > M^{3d/2} \approx 2^{3d\rho/2}$

i.e. $\lambda \approx 3d\rho/2$

and $\eta \approx \dfrac{\lambda^2}{\rho} - \lambda = \dfrac{3d\lambda}{2} - \lambda = \dfrac{3d\rho}{2}\left(\dfrac{3d}{2} - 1\right)$

If $n \approx \sqrt{M}$, $M \approx 2^{\rho}$ then we may take $\lambda \approx 3d\rho/2$ and $\eta \approx 3d\lambda/2 - \lambda$. For, example, if $\rho = 32$, $d = 4$, we can take any $\lambda > 192$, $\eta > 960$.

Note that $\lambda$ scales linearly with $d$ and $\eta$ scales quadratically. These bounds carry over to HE2 and HE$k$.

*3.1.3 Encryption*

We encrypt a plaintext integer $m$ using Enc (Algorithm 3).

**Algorithm 3:** Enc: Encryption algorithm

**Input** : $m \in \mathcal{M}$
**Input** : $p$: secret key
**Input** : modulus: public modulus
**Output:** $c \in \mathcal{C}$
1 $q \leftarrow$ modulus$/p$
2 $r \leftarrow_{\$} [1, q)$
3 $c \leftarrow m + rp \pmod{\text{modulus}}$
4 **return** $c$

*3.1.4 Decryption*

We decrypt the ciphertext $c$ using Dec (Algorithm 4).

**Algorithm 4:** Dec: Decryption algorithm

**Input** : $c \in \mathcal{C}$
**Input** : $p$: secret key
**Output:** $m \in \mathcal{M}$
1 $m \leftarrow c \pmod{p}$
2 **return** $m$

*3.1.5 Addition*

The sum modulo $pq$ of two ciphertexts, $c = m + rp$ and $c' = m' + r'p$, is given by Add (Algorithm 5). Since

**Algorithm 5:** Add: addition algorithm

**Input** : $c \in \mathcal{C}$
**Input** : $c' \in \mathcal{C}$
**Input** : modulus $\in \mathbb{Z}$: public modulus
**Output:** result $\in \mathcal{C}$
1 result $\leftarrow c + c' \pmod{\text{modulus}}$
2 **return** result

$\mathsf{Add}(c, c') = c + c' = m + m' + (r + r')p$, $\mathsf{Add}(c, c')$ decrypts to $m + m'$, provided $m + m' < p$.

*3.1.6 Multiplication*

The product modulo $pq$ of two ciphertexts, $c = m + rp$ and $c' = m' + r'p$, is given by Mult (Algorithm 6). Since $\mathsf{Mult}(c, c') = cc' = mm' + (rm' + r'm + rr'p)p$, it

**Algorithm 6:** Mult: multiplication algorithm

**Input** : $c \in \mathcal{C}$
**Input** : $c' \in \mathcal{C}$
**Input** : modulus $\in \mathbb{Z}$: public modulus
**Output:** result $\in \mathcal{C}$
1 result $\leftarrow cc' \pmod{\text{modulus}}$
2 **return** result

decrypts to $mm'$, provided $mm' < p$.

*3.1.7 Security*

Security of the system is provided by the *partial approximate common divisor problem* (PACDP), first posed by Howgrave-Graham [50], but can be formulated [24, 28] as:

**Definition 1 (Partial approximate common divisor problem.)** Suppose we are given one input $x_0$, of the form $pr_0$, and $n$ inputs $x_i$, of the form $pr_i + m_i$, $i \in [1, n]$, where $p$ is an unknown constant integer and the $m_i$ and $r_i$ are unknown integers. We have a bound

$B$ such that $|m_i| < B$ for all $i$. Under what conditions on the $m_i$ and $r_i$, and the bound $B$, can an algorithm be found that can uniquely determine $p$ in time polynomial in the total bit length of the numbers involved?

A straightforward attack on this problem is by brute force. Consider $x_1$. Assuming that $m_1$ is sampled from $\mathcal{D}$, having entropy $\rho$, we successively try values for $m_1$ and compute $\gcd(x_0, x_1 - m_1)$ in polynomial time until we find a divisor that is large enough to recover $p$. Then we can recover $m_i$ as $(x_i \bmod p)$ for $i \in [2, n]$. As discussed in section 2.3, the search will requires $2^\rho$ gcd operations in expectation. Note that publicly known constants, need not, and should not be encrypted. Encrypting them provides an obvious guessing attack.

Some attempts have been made to solve the PACDP [24,28,50], resulting in theoretically faster algorithms for some cases of the problem. The paper [24] gives an algorithm requiring only $\sqrt{M}$ polynomial time operations if $\mathcal{D}$ is the uniform distribution on $[0, M)$, and hence $\rho = \lg M$. No algorithm running in time subexponential in $\rho$ is known for this problem, so the encryption will be secure if $\rho$ is large enough. See [43] for a survey and evaluation of attacks on PACDP. We also note the work of Cheon and Stehlé [27] which shows that Regev's "learning with errors" (LWE) problem [68] can be reduced to the *approximate common divisor problem* (ACDP), demonstrating that ACDP is at least as hard as LWE. LWE is the basis of many lattice based FHE schemes.

Our system is a special case of PACDP, since we use the residues modulo a distinct semiprime. A semiprime is a natural number that is the product of two primes. A distinct semiprime is a semiprime where the primes are distinct. We call this the *semiprime partial approximate common divisor problem* (SPACDP). It is a restriction, but there is no reason to believe that it is any easier than PACDP.

**Definition 2 (Semiprime factorisation problem.)** Given a semiprime $s$, the product of primes $p$ and $q$, can $p$ and $q$ be determined in polynomial time?

The computational complexity of this problem, which lies at the heart of the widely-used RSA cryptosystem, is open, other than for quantum computing, which currently remains impractical. Strong semiprimes, as are used here, are generally believed to be the hardest to factor. We will show that breaking HE1 is equivalent to semiprime factorisation. Therefore, our scheme is at least as secure as unpadded RSA [71].

**Theorem 1** *An attack against HE1 is successful in polynomial time if and only if we can factorise a distinct semi-prime in polynomial time.*

*Proof* Suppose that we have an unknown plaintext $m$, encrypted as $c = m + rp \mod pq$, where $r \leftarrow_\$ [1, q)$.

If we can factor $pq$ in polynomial time, we can determine $p$ and $q$ in polynomial time, since we know $p < q$. Therefore, we can determine $m = c \bmod p$.

If we can determine $m$ given $c$ for arbitrary $m$, then we can determine $rp = c - m$. We are given $qp$, and we know $0 < r < q$, so $\gcd(rp, qp)$ must be $p$, and we can compute $p$ in polynomial time. Now, given $p$, we can determine $q$ as $qp/p$. Hence, we can factorise $pq$ in polynomial time. □

With low entropy plaintexts, there is a brute force attack on this system, which we call a *collision attack*. Suppose we have a pair of equal plaintexts $m_1 = m_2$. The difference between their encryptions $(c_1 - c_2)$ is an encryption of 0, and KPA is possible. In fact, for $n$ plaintexts $m_1, m_2, \ldots, m_n$, if there exist $i, j \in [1, n]$ with $m_i = m_j$, then $\prod_{1 \le i < j \le n}(c_j - c_i)$ is an encryption of 0. However, if there is sufficient entropy, this attack is not possible.

**Lemma 1** *If the inputs $m$ have entropy $\rho$ then, for any two independent inputs $m_1, m_2$, $\Pr(m_1 = m_2) \le 2^{-\rho}$.*

*Proof* $\Pr(m_1 = m_2) = \sum_{i=0}^{M-1} \xi_i^2 = 2^{-H_2} \le 2^{-\rho}$, since $H_2 \ge H_\infty = \rho$. □

Thus, for $n$ inputs, $m_1, m_2, \ldots, m_n$ the probability that there exist $i, j \in [1, n]$ with $m_i = m_j$ is at most $\binom{n}{2} 2^{-\rho}$. If $n < 2^{-\rho/3}$, this probability is at most $2^{-\rho/3}$. Hence, for large enough $\lambda$, collision attack is infeasible.

### 3.2 Insufficient Entropy (HE1N)

Suppose now that the integer inputs $m_i, i \in [1, n]$, are distributed with entropy $\rho$, where $\rho$ is not large enough to negate a brute force guessing attack. Therefore, we increase the entropy of the plaintext by adding an additional "noise" term to the ciphertext. This will be a multiple $s$ (from 0 to $\kappa$) of an integer $\kappa$, chosen so that the entropy $\rho' = \rho + \lg \kappa$ is large enough to negate a brute force guessing attack. As a result of the extra linear term in the ciphertext, we compute the quantity $P(m_1, \ldots, m_n, \kappa)$ instead. We can easily retrieve $P(m_1, \ldots, m_n)$ from $P(m_1, \ldots, m_n, \kappa)$.

#### *3.2.1 Key and Parameter Generation*

KGen (Algorithm 7) and Pgen(Algorithm 7) now randomly choose $p$ and $q$ as in HE1, but with $\eta = \lambda^2/\rho' - \lambda$, and $p > (n+1)^d (M + \kappa^2)^d$ so that $P(m_1 + s_1\kappa, m_2 + s_2\kappa, \ldots, m_N + s_n\kappa) < p$, when $s_1, s_2, \ldots, s_n \in [0, \kappa)$. KGen also randomly chooses $\kappa$, where $\kappa > (n+1)^d M^d$, so that $P(m_1, m_2, \ldots, m_n) < \kappa$. The secret key, sk, is now $(\kappa, p)$.

**Algorithm 7:** KGen: Key Generation Algorithm

**Input** : $\lambda \in \mathcal{S}$
**Input** : $\rho \in \mathbb{Z}$: entropy of input
**Input** : $\rho' \in \mathbb{Z}$: effective entropy of inputs
**Output:** $(\kappa, p)$: secret key
$p \leftarrow_{\mathsf{p}} [2^{\lambda-1}, 2^{\lambda}]$
$\nu \leftarrow \rho' - \rho$
$\kappa \leftarrow_{\mathsf{p}} [2^{\nu-1}, 2^{\nu}]$
**return** $(\kappa, p)$

**Algorithm 8:** Pgen: Parameter Generation Algorithm

**Input** : $\lambda \in \mathcal{S}$
**Input** : $\rho' \in \mathbb{Z}$: effective entropy of inputs
**Input** : $(\kappa, p)$: secret key
**Output:** modulus $\in \mathbb{Z}$: modulus for arithmetic
$\eta \leftarrow \lambda^2/\rho' - \lambda$
$q \leftarrow_{\mathsf{p}} [2^{\eta-1}, 2^{\eta}]$
modulus $\leftarrow pq$
**return** modulus

*3.2.2 Encryption*

We encrypt plaintext $m$ using Enc(Algorithm 9).

**Algorithm 9:** Enc: Encryption Algorithm

**Input** : $m \in \mathcal{M}$
**Input** : $(\kappa, p)$: secret key
**Input** : modulus: public modulus
**Output:** $c \in \mathcal{C}$
$q \leftarrow$ modulus$/p$
$r \leftarrow_{\$} [1, q)$
$s \leftarrow_{\$} [0, \kappa)$
$c \leftarrow m + s\kappa + rp \pmod{\text{modulus}}$
**return** $c$

*3.2.3 Decryption*

We decrypt ciphertext $c$ using Dec(Algorithm 10).

**Algorithm 10:** Dec: Decryption Algorithm

**Input** : $c \in \mathcal{C}$
**Input** : $(\kappa, p)$: secret key
**Output:** $m \in \mathcal{M}$
$m \leftarrow (c \bmod p) \bmod \kappa$
**return** $m$

*3.2.4 Arithmetic*

Addition and multiplication of ciphertexts is given by Algorithms 5 and 6.

*3.2.5 Security.*

The use of random noise gives the encryption the following "indistinguishability" property, which we will use to show that HE1N satisfies IND-CPA [6,7].

**Lemma 2** *For any encryption $c$, $c \bmod \kappa$ is polynomial time indistinguishable from the uniform distribution on $[0, \kappa)$.*

*Proof* Let $c = m + s\kappa + rp = m + rp \mod \kappa$, where $r \leftarrow_{\$} [1, q)$. Then, for $i \in [0, \kappa)$,

$$\begin{aligned}\Pr\left(c \bmod \kappa = i\right) &= \Pr(m + rp = i \mod \kappa) \\ &= \Pr\left(r = p^{-1}(i - m) \mod \kappa\right) \\ &\in \left\{\lfloor q/\kappa \rfloor 1/q, \lceil q/\kappa \rceil 1/q\right\} \\ &\in [1/\kappa - 1/q, 1/\kappa + 1/q],\end{aligned}$$

where the inverse $p^{-1}$ of $p \bmod \kappa$ exists since $p$ is a prime. Hence the total variation distance from the uniform distribution is

$$\tfrac{1}{2}\sum_{i=0}^{\kappa-1} |\Pr\left(c \bmod \kappa = i\right) - 1/\kappa| < \kappa/q.$$

This is exponentially small in the security parameter $\lambda$ of the system, so the distribution of $c \bmod \kappa$ cannot be distinguished in polynomial time from the uniform distribution. □

We can further show that an adversary learns nothing about the plaintexts from the size of the corresponding ciphertexts.

**Lemma 3** *The ciphertexts $c_1$,$c_2$ for any plaintexts $m_1$, $m_2$ satisfy*

$$\Pr(c_1 \geq c_2) \leq \frac{1}{2} \pm \frac{1}{2q}.$$

*Proof*

$$\begin{aligned}\Pr(c_1 \geq c_2) &= \Pr(m_1 + s_1\kappa + r_1 p \geq m_2 + s_2\kappa + r_2 p) \\ &\leq Pr(r_1 p \geq r_2 p - \kappa^2) \\ &= Pr(r_1 \geq r_2 - {}^{\kappa^2}/_{p}) \\ &= Pr(r_1 \geq r_2), \text{ since } {}^{\kappa^2}/_{p} < 1 \\ &= \sum_{i=1}^{q} \frac{i}{q} \cdot \frac{1}{q} = \frac{q(q+1)}{q^2} = \frac{1}{2} + \frac{1}{2q}\end{aligned}$$

Similarly,

$$\Pr(c_2 \geq c_1) \leq \frac{1}{2} + \frac{1}{2q}$$
$$\implies \Pr(c_1 \geq c_2) \leq \frac{1}{2} - \frac{1}{2q}$$

So,

$$\Pr(c_1 \geq c_2) \leq \frac{1}{2} \pm \frac{1}{2q}$$

□

We now proceed to the proof of IND-CPA. First, we define the *general approximate common divisor problem* (GACDP)[50,24,28].

**Definition 3 (General approximate common divisor problem.)** Suppose we are given $n$ inputs $x_i$, of the form $pr_i+m_i$, $i \in [1,n]$, where $p$ is an unknown constant integer and the $m_i$ and $r_i$ are unknown integers. We have a bound $B$ such that $|m_i| < B$ for all $i$. Under what conditions on the $m_i$ and $r_i$, and the bound $B$, can an algorithm be found that can uniquely determine $p$ in time polynomial in the total bit length of the numbers involved?

**Theorem 2** *HE1N satisfies IND-CPA[6], under the assumption that GACDP is not polynomial time solvable.*

*Proof* Suppose that known plaintexts $m_1, \ldots, m_n$ are encrypted by an oracle for HE1N, producing ciphertexts $c_1, \ldots, c_n$. Then, for $r_i \leftarrow_\$ [0,q)$, $s_i \leftarrow_\$ [0,\kappa)$, we have an SPACDP with ciphertexts $c_i = m_i + s_i\kappa + r_ip$, and the approximate divisor $p$ cannot be determined in polynomial time in the worst case using non-quantum methods. However, the offsets in this SPACDP are all of the form $m_i + s_i\kappa$, for known $m_i$, and we must make sure this does not provide information about $p$. To show this, we rewrite the SPACDP as

$$c_i = m_i + s_i\kappa + r_ip = m'_i + s'_i\kappa, \quad (i = 1,2,\ldots,n), \quad (1)$$

where $s'_i = s_i + \lfloor (m_i + r_ip)/\kappa \rfloor$, and $m'_i = m_i + r_ip \pmod{\kappa}$. Now we may view (1) as a GACDP, with "encryptions" $m'_i$ of the $m_i$, and approximate divisor $\kappa$. Since the offsets $m'_i$ are polynomial time indistinguishable from uniform $[0,\kappa)$, from Lemma 2, we will not be able to determine $\kappa$ in polynomial time. Now, the offsets $m'_1, m'_2$ of any two plaintexts $m_1, m_2$ are polynomial time indistinguishable from $m'_2, m'_1$, since they are indistinguishable from two independent samples from uniform $[0,\kappa)$. Therefore, in polynomial time, we will not be able to distinguish between the encryption $c_1$ of $m_1$ and the encryption $c_2$ of $m_2$.

Therefore, if $\mathcal{A}$ is a polynomial time adversary, from Lemmas 2 and 3 the advantage of $\mathcal{A}$, is:

$$\mathsf{Adv}^{\text{ind-cpa}}_{\mathcal{A},HE1N}(\lambda) \leq 2\left(\frac{\kappa}{q} + \frac{1}{2q}\right) - 1 = \frac{2\kappa+1}{q} = \mathsf{negl}(\lambda)$$

□

Therefore, HE1N is resistant to both the "guessing" and "collision" attacks discussed in section 3.1.

**Lemma 4** *Decrypting HE1N without knowledge of $\kappa$ is polynomial time Turing equivalent to GACDP.*

*Proof* Suppose we can solve GACDP instances in polynomial time. If $p$ is unknown, we have an instance of GACDP with offsets $m_i + s_i\kappa$ and approximate divisor $p$. We can solve this to obtain the offsets $m_i + s_i\kappa$. These offsets form a new instance of GACDP with offset $m_i$ and approximate divisor $\kappa$. We can then solve this to obtain $m_i$. If $p$ is known, we need only solve the $m_i + s_i\kappa$ GACDP instance. Hence, we can decrypt any ciphertext.

Conversely, suppose we can decrypt any HE1N system in polynomial time. Then, if $t_i = m_i + s_i\kappa$ is an instance of GACDP with approximate divisor $\kappa$, we choose $p \geq q \geq \max_i m_i$ and generate ciphertexts $c_i$. Then the decryption algorithm for HE1N will recover the $m_i$. Then $\kappa$ can be recovered using gcd on the $(t_i - m_i)$ and we can solve the GACDP instance. □

Therefore, as a result of the additional term, $\kappa$, HE1N is quantum resistant [11] despite the public semiprime modulus $pq$.

### *3.2.6 Hybrid scheme*

Note that mixed data, some of which has high entropy and some low, can be encrypted with a hybrid of HE1 and HE1N. More generally, we can choose $s$ to be smaller for higher entropy and larger for lower entropy, say $s \in [0,\chi_i)$, where $0 \leq \chi_i < \kappa$, for the $i$th data type, rather than $[0,\kappa)$. However, $\kappa$ itself remains the same for all $i$, or we cannot decrypt. Then the entropy increases to $\rho_i + \lg \chi_i$ for data type $i$. The advantage is a smaller blow-up in the noise. A possible disadvantage is that this mixed scheme may not necessarily have the IND-CPA property of Theorem 2. The same idea can be applied to HE2 and HE2N below, and to the HE$k$N schemes, for $k > 2$, described in section 5.2.

### *3.2.7 Noise Growth*

So that the "noise" terms of the ciphertext do not grow sufficiently large to cause the computational result to

overflow the secret modulus $p$, we need to control the growth of these terms. As a we are computing a polynomial of degree $d$, each multiplication doubles the bit length of the noise terms and each addition adds at most one bit.

To avoid overflow, as stated earlier, we set $p > (n+1)^d(M+\kappa^2)^d$ so that $P(m_1 + s_1\kappa, m_2 + s_2\kappa, \ldots, m_N + s_n\kappa) < p$. This is a worst case bound, based on the largest possible coefficient of any term. Depending on the particular polynomial to compute, a lower bound on $p$ may suffice to avoid overflow.

#### *3.2.8 Security parameters*

Again, we can set the security parameters $\lambda$ and $\eta$ to practical values. For HE1N, we assume $M \approx 2^\rho$, and we have $\rho' = \rho + \lg\kappa$. Now,

$$\kappa > (n+1)^d M^d \approx (nM)^d \text{ for large } n,$$

$$\text{i.e. } \lg\kappa \approx d(\lg n + \rho)$$

Therefore, since $\rho = \rho' - \lg\kappa$,

$$\lg\kappa > d\lg n + d(\rho' - \lg\kappa)$$

$$\text{i.e. } \lg\kappa \approx \frac{d(\lg n + \rho')}{d+1}$$

Since $\kappa$ is much larger than $M$, we also have

$$p = 2^\lambda > (n+1)^d(M+\kappa^2)^d \approx (n\kappa^2)^d \text{ for large } n$$

$$\text{i.e. } \lambda \approx d(\lg n + 2\lg\kappa),$$

$$\text{and } \eta \approx \frac{\lambda^2}{\rho'} - \lambda = \frac{3d\lambda}{2} - \lambda = \frac{3d\rho'}{2}\left(\frac{3d}{2} - 1\right)$$

Then we can calculate $\eta$ as for HE1 above. Again, $\lambda$ scales linearly with $d$ and $\eta$ scales quadratically. These bounds carry over to HE2N and HE$k$N.

If we assume $M \approx 2^\rho$ and large enough $n$, as in section 3.1, then we may take $\lg\kappa > d(\lg n + \rho)$, $\rho' = \rho + \lg\kappa$, $\lambda > d(\lg n + 2\lg\kappa)$. Then, for example, if $d = 3$, $\lg n = 16$, $\rho = 8$, then $\lg\kappa > 72$, $\rho' = 80$, $\lambda > 480$, $\eta > 2400$. In the extreme case that the inputs are bits, so $\rho = 1$, and $d = 3$, $\lg n = 16$, then we can take $\lg\kappa \approx 51$ and $\rho' \approx 52$, and we have $\lambda > 354$, $\eta > 2056$, which is only 15% smaller than for $\rho = 8$.

We must also ensure that $\kappa$ is large enough to make the GACDP instances hard to solve (see [24,28,50] for details). Note that $\kappa \ll q$ for Theorem 2 to hold.

## 4 Adding a dimension

In this section we discuss adding an additional dimension to the ciphertext, which becomes a 2-vector. The purpose of this is to increase the level of security beyond HE1 and HE1N. In both schemes presented below, HE2 and HE2N, we add a further vector term, with two further secret parameters. The two schemes presented below have a constant factor overhead for arithmetic operations. An addition operation in the plaintext space requires two additions in the ciphertext space, and a multiplication in the plaintext space requires nine multiplications and four additions in the ciphertext space.

### 4.1 Sufficient entropy (HE2)

As noted above, in this scheme we now add a multiple of a secret vector, $\mathbf{a} = [a_1 a_2]^T$, to the ciphertext. Therefore, we encrypt $m$ as

$$\mathbf{c} = (m + rp)\mathbf{1} + s\mathbf{a},$$

where $\mathbf{1}$ is the vector $[1\ 1]^T$ and $s \leftarrow_\$ [0, pq)$. As with HE1, it is assumed that the inputs $m_i$ ($i \in [1, n]$) are of sufficient entropy.

We decrypt $c$ by eliminating $a_1$ and $a_2$ from the ciphertext and then taking the residue modulo $p$, i.e.

$$m = \boldsymbol{\gamma}^T\mathbf{c} \mod p,$$

where $\boldsymbol{\gamma}^T = (a_2 - a_1)^{-1}[a_2 \ \ -a_1]$.

We can easily see that this construction is homomorphic over addition. For HE2, we now define multiplication of ciphertexts, $\mathbf{c}_1$ and $\mathbf{c}_2$ as the Hadamard (elementwise) product of the augmented ciphertext vectors $\mathbf{c}_{1\star}$ and $\mathbf{c}_{1\star}$ where the augmentation of $\mathbf{c} = [c_1\ c_2]^T$ is given by $\mathbf{c}_\star = [c_1\ c_2\ c_3]^T$, where $c_3 = 2c_1 - c_2$.

However, this Hadamard product will include additional $a_1^2$ and $a_2^2$ terms which means it is not a valid ciphertext. Therefore we re-encrypt by applying a matrix $R$ to the Hadamard product to eliminate these quadratic terms. The construction of $R$ is detailed in sections 4.1.1 and 4.1.5.

#### *4.1.1 Key and Parameter Generation*

$p$ and $q$ are randomly chosen by KGen (Algorithm 11) according to the bounds given in section 3.1. KGen sets $\mathbf{a} = [a_1\ a_2]^T$, where $a_i \leftarrow_\$ [1, pq)$ ($i \in [1, 2]$) such that $a_1, a_2, a_1 - a_2 \neq 0 \pmod p$ and $\bmod q$).[3]

Pgen(Algorithm 12) generates $R$, the re-encryption matrix.

#### *4.1.2 Encryption*

We encrypt a plaintext integer $m$ as the 2-vector $\mathbf{c}$ using Enc(Algorithm 13), where $\mathbf{1} = [1\ 1]^T$ $r$ and $s$ are

[3] The condition $a_1, a_2, a_1 - a_2 \neq 0$, (mod $p$, mod $q$) fails with exponentially small probability $3(1/p + 1/q)$. Thus, $a_1$ and $a_2$ are indistinguishable in polynomial time from $a_1, a_2 \leftarrow_\$ [0, pq)$.

**Algorithm 11:** KGen: Key Generation Algorithm

**Input** : $\lambda \in \mathcal{S}$
**Input** : $\rho \in \mathbb{Z}$: entropy of inputs
**Output:** $(p, \mathbf{a})$: secret key
**Output:** modulus: public modulus
$p \leftarrow_{\mathsf{p}} [2^{\lambda-1}, 2^{\lambda}]$
$\eta \leftarrow \lambda^2/\rho - \lambda$
$q \leftarrow_{\mathsf{p}} [2^{\eta-1}, 2^{\eta}]$
modulus $\leftarrow pq$
**repeat**
  $a_i \leftarrow_{\$} [1, \text{modulus})$ $(i \in [1, 2])$
**until** $a_1, a_2, a_1 - a_2 \neq 0$ *(*mod $p$ *and* mod $q$*)*
$\mathbf{a} \leftarrow [a_1 \ a_2]^T$
**return** $(p, \mathbf{a})$, modulus

**Algorithm 12:** Pgen: Parameter Generation Algorithm

**Input** : $(p, \mathbf{a})$: secret key
**Input** : modulus: public modulus
**Output:** $R$: public re-encryption matrix
$q \leftarrow \text{modulus}/p$
$\beta \leftarrow 2(a_2 - a_1)^2$
$\varrho \leftarrow_{\$} [0, q)$
$\sigma \leftarrow_{\$} [0, \text{modulus})$
$\alpha_1 \leftarrow \beta^{-1}(\sigma a_1 + \varrho p - a_1^2)$
$\alpha_2 \leftarrow \beta^{-1}(\sigma a_2 + \varrho p - a_2^2)$
$R \leftarrow \begin{bmatrix} 1 - 2\alpha_1 & \alpha_1 & \alpha_1 \\ -2\alpha_2 & \alpha_2 + 1 & \alpha_2 \end{bmatrix}$
**return** $R$

**Algorithm 13:** Enc: Encryption Algorithm

**Input** : $m \in \mathcal{M}$
**Input** : $(p, \mathbf{a})$: secret key
**Input** : modulus: public modulus
**Output:** $\mathbf{c} \in \mathcal{C}$
$q \leftarrow \text{modulus}/p$
$r \leftarrow_{\$} [0, q)$
$s \leftarrow_{\$} [0, \text{modulus})$
$\mathbf{c} \leftarrow (m + rp)\mathbf{1} + s\mathbf{a} \pmod{\text{modulus}}$
**return** $\mathbf{c}$

independent. We note that two encryptions of the same plaintext are different with very high probability.

**Theorem 3** *The encryption scheme produces ciphertexts with components which are random integers modulo pq.*

*Proof* Consider a ciphertext vector which encrypts the plaintext, $m$, and the expression $m + rp + sa \mod pq$ which represents one of its elements. Then $r \leftarrow_{\$} [0, q)$, $s \leftarrow_{\$} [0, pq)$.

Consider first $m + sa$. We know that $a^{-1} \mod pq$ exists because $a \neq 0 \pmod p$ and mod $q$). Thus, conditional on $r$,

$$\Pr[m + rp + sa = i \bmod pq] =$$
$$\Pr[s = a^{-1}(i - m - rp) \bmod pq] = \frac{1}{pq}.$$

Since this holds for any $i \in [0, pq)$, $m + ra + sp \mod pq$ is a uniformly random integer from $[0, pq)$. □

Note, however, that the components of the ciphertexts are correlated, and this may be a vulnerability. We discuss this later in this section ("Cryptanalysis").

### *4.1.3 Decryption*

To decrypt, we use Algorithm 14. We call $\boldsymbol{\gamma}$ the *decryption vector*.

**Algorithm 14:** Dec: Decryption Algorithm

**Input** : $\mathbf{c} \in \mathcal{C}$
**Input** : $(p, \mathbf{a})$: secret key
**Output:** $m \in \mathcal{M}$
$\boldsymbol{\gamma}^T \leftarrow (a_2 - a_1)^{-1}[a_2 \ \ -a_1]$
$m \leftarrow \boldsymbol{\gamma}^T \mathbf{c} \mod p$
**return** $m$

### *4.1.4 Addition*

We define the addition operation on ciphertexts as the vector sum modulo $pq$ of the two ciphertext vectors $\mathbf{c}$ and $\mathbf{c}'$.

**Algorithm 15:** Add: addition algorithm

**Input** : $\mathbf{c} \in \mathcal{C}$
**Input** : $\mathbf{c}' \in \mathcal{C}$
**Input** : modulus $\in \mathbb{Z}$: modulus for arithmetic
**Output:** result $\in \mathcal{C}$
result $\leftarrow \mathbf{c} + \mathbf{c}' \pmod{\text{modulus}}$
**return** result

Therefore, if inputs $m, m'$ encrypt as $(m+rp)\mathbf{1}+s\mathbf{a}$, $(m' + r'p+)\mathbf{1} + s'\mathbf{a}$,

$$\mathsf{Add}(\mathbf{c}, \mathbf{c}') = \mathbf{c} + \mathbf{c}' = (m + m' + (r + r')p)\mathbf{1} + (s + s')\mathbf{a}.$$

which is a valid encryption of $m + m'$.

### *4.1.5 Multiplication*

If $\mathbf{c} = [c_1 \ c_2]^T$, we construct the augmented ciphertext vector, $\mathbf{c}_\star = [c_1 \ c_2 \ c_3]^T$, where $c_3 = 2c_1 - c_2$. Thus, $c_3 = (m + rp) + sa_3 \mod pq$, for $a_3 = 2a_1 - a_2$.

Now, consider the Hadamard product modulo $pq$, $\mathbf{c}_\star \circ \mathbf{c}'_\star$, of the two augmented ciphertext vectors $\mathbf{c}_\star$ and $\mathbf{c}_\star{}'$:

$$\mathbf{z}_\star = \mathbf{c}_\star \circ \mathbf{c}'_\star = \begin{bmatrix} c_1 c'_1 \\ c_2 c'_2 \\ c_3 c'_3 \end{bmatrix} \mod pq$$

Therefore, if inputs $m, m'$ are encrypted as $(m + rp)\mathbf{1} + s\mathbf{a}$, $(m' + r'p)\mathbf{1} + s'\mathbf{a}$, we first calculate

$$\begin{aligned}\mathbf{z}_\star &= (m + rp)(m' + r'p)\mathbf{1}_\star \\ &\quad + [(m + rp)s' + (m' + r'p)s]\mathbf{a}_\star + ss'\mathbf{a}_\star^{\circ 2} \\ &= (mm' + r_1 p)\mathbf{1}_\star + s_1\mathbf{a}_\star + ss'\mathbf{a}_\star^{\circ 2} \mod pq,\end{aligned}$$

where $r_1 = mr' + m'r + rr'p$, $s_1 = (m + rp)s' + (m' + r'p)s$, and $\mathbf{a}_\star^{\circ 2} = [a_1^2 \;\; a_2^2 \;\; a_3^2]^T$.

As we can see, $\mathbf{z}_\star$ is not a valid encryption of $mm'$. We need to re-encrypt this product to eliminate the $\mathbf{a}_\star^{\circ 2}$ term.

We achieve this by multiplying $\mathbf{z}_\star$ by $R$. It is easy to check that $R\mathbf{1}_\star = \mathbf{1}$ and $R\mathbf{a}_\star = \mathbf{a}$, independently of $a_1, a_2$. Now

$$\begin{aligned}(R\mathbf{a}_\star^{\circ 2})_1 &= (1 - 2\alpha_1)a_1^2 + \alpha_1 a_2^2 + \alpha_1(2a_1 - a_2)^2 \\ &= a_1^2 + \alpha_1((2a_1 - a_2)^2 + a_2^2 - 2a_1^2) \\ &= a_1^2 + 2\alpha_1(a_2 - a_1)^2 \\ &= a_1^2 + \alpha_1\beta \\ &= \varrho p + \sigma a_1 \\ (R\mathbf{a}_\star^{\circ 2})_2 &= -2\alpha_2 a_1^2 + (\alpha_2 + 1)a_2^2 + \alpha_2(2a_1 - a_2)^2 \\ &= a_2^2 + \alpha_2((2a_1 - a_2)^2 + a_2^2 - 2a_1^2) \\ &= a_2^2 + 2\alpha_2(a_2 - a_1)^2 \\ &= a_2^2 + \alpha_2\beta \\ &= \varrho p + \sigma a_2\end{aligned}$$

Thus, we obtain the identity $R\mathbf{a}_\star^{\circ 2} = \varrho p\mathbf{1} + \sigma\mathbf{a}$.

So, applying $R$ to $\mathbf{z}_\star$, i.e. $\mathbf{z}' = R\mathbf{z}_\star$, gives

$$\begin{aligned}\mathbf{z}' &= (mm' + r_1 p)R\mathbf{1} + s_1 R\mathbf{a} + ss' R\mathbf{a}^{\circ 2} \\ &= (mm' + r_1 p)\mathbf{1} + s_1\mathbf{a} + ss'(\sigma\mathbf{a} + \varrho p\mathbf{1}) \\ &= (mm' + r_2 p)\mathbf{1} + (s_1 + \sigma rr')\mathbf{a} \\ &= (mm' + r_2 p)\mathbf{1} + s_2\mathbf{a} \qquad (\mathrm{mod}\ pq)\end{aligned}$$

for some integers $r_2, s_2$. So $\mathbf{z}'$ is a valid encryption of $mm'$ and our homomorphic multiplication operation on ciphertexts is $R(\mathbf{c}_\star \circ \mathbf{c}'_\star)$.

$$\mathsf{Mult}(\mathbf{c}, \mathbf{c}') = \mathbf{c} \cdot \mathbf{c}' = R(\mathbf{c}_\star \circ \mathbf{c}'_\star) \pmod{pq},$$

where $\cdot$ is a product on $\mathbb{Z}_{pq}^2$ and $\mathbf{c}_\star \circ \mathbf{c}'_\star$ is the Hadamard product modulo $pq$ of the two augmented ciphertext vectors $\mathbf{c}_\star$ and $\mathbf{c}'_\star$ (see Algorithm 16).

Observe that $\alpha_1, \alpha_2$ in $R$ are public, but give only two equations for the four parameters of the system $a_1, a_2, \sigma, \varrho p$. These equations are quadratic mod $pq$, so solving them is as hard as semiprime factorisation in the worst case [66].

Also, observe that, independently of $\mathbf{a}$,

$$R\mathbf{c}_\star = (m + rp)R\mathbf{1}_\star + sR\mathbf{a}_\star = (m + rp)\mathbf{1} + s\mathbf{a} = \mathbf{c},$$

for any ciphertext $\mathbf{c}$. Hence re-encrypting a ciphertext gives the identity operation, and discloses no information.

**Algorithm 16:** Mult: multiplication algorithm

**Input** : $\mathbf{c} = [c_1 \; c_2]^T \in \mathcal{C}$
**Input** : $\mathbf{c}' = [c'_1 \; c'_2]^T \in \mathcal{C}$
**Input** : modulus $\in \mathbb{Z}$: public modulus
**Input** : $R$: re-encryption matrix
**Output:** result $\in \mathcal{C}$

$c_3 \leftarrow 2c_1 - c_2$
$\mathbf{c}_\star \leftarrow [c_1 \; c_2 \; c_3]^T$
$c'_3 \leftarrow 2c'_1 - c'_2$
$\mathbf{c}'_\star \leftarrow [c'_1 \; c'_2 \; c'_3]^T$
result $\leftarrow R(\mathbf{c}_\star \circ \mathbf{c}'_\star)$ (mod modulus)
**return** result

*4.1.6 Hardness*

We can show that this system is at least as hard as SPACDP. In fact,

**Theorem 4** *SPACDP is of equivalent complexity to the special case of HE2 where $\delta = a_2 - a_1$ $(0 < \delta < p)$ is known.*

*Proof* Suppose we have a system of $n$ approximate multiples of a prime $p$, $m_i + r_i p$ $(i = 1, 2, \ldots, n)$. Then we generate values $a, s_1, s_2, \ldots, s_n \leftarrow_\$ [0, pq)$, and we have an oracle set up the cryptosystem with $a_1 = a$, $a_2 = a + \delta$. The oracle has access to $p$ and provides us with $R$, but no information about its choice of $\varrho$ and $\sigma$. We then generate the ciphertexts $\mathbf{c}_i$ $(i = 1, 2, \ldots, n)$:

$$\begin{bmatrix} c_{i1} \\ c_{i2} \end{bmatrix} = \begin{bmatrix} m_i + r_i p + s_i a \\ m_i + r_i p + s_i(a + \delta) \end{bmatrix} \pmod{pq}. \tag{2}$$

Thus $c_{i1} - s_i a = c_{i2} - s_i(a + \delta) = m_i + r_i p$. Thus finding the $m_i$ in (2) in polynomial time solves SPACDP in polynomial time.

Conversely, suppose we have any HE2 system with $a_2 = a_1 + \delta$. The ciphertext for $m_i$ $(i = 1, 2, \ldots, n)$ is as in (2). So $s_i = \delta^{-1}(c_{i2} - c_{i1})$. Since $0 < \delta < p < q$, $\delta$ is coprime to both $p$ and $q$, and hence $\delta^{-1} \mod pq$ exists. Thus breaking the system is equivalent to determining the $m_i \mod p$ from $m_i + \delta^{-1}(c_{i2} - c_{i1})a + r_i p$

$(i = 1, 2, \ldots, n)$. Determining the $m_i + \delta^{-1}(c_{i2} - c_{i1})a$ from the $m_i + \delta^{-1}(c_{i2} - c_{i1})a + r_i p$ $(i = 1, 2, \ldots, n)$ can be done using SPACDP. However, we still need to determine $a$ in order to determine $m_i$. This can be done by "deciphering" $R$ using SPACDP. We have

$$2\delta^2\alpha_1 = \sigma a - a^2 + \varrho p, \qquad 2\delta^2\alpha_2 = \sigma(a+\delta) - (a+\delta)^2 + \varrho p,$$

so $\sigma = 2\delta^2(\alpha_2 - \alpha_1) - 2\delta a - \delta^2$. Now $a$ can be determined by first determining $m_0 = a(2\delta^2(\alpha_2 - \alpha_1) - (2\delta + 1)a - \delta^2)$ from $m_0 + \varrho p = 2\delta^2\alpha_1$. This can be done using SPACDP. Then $a$ can be determined by solving the quadratic equation $m_0 = a(2\delta^2(\alpha_2 - \alpha_1) - (2\delta + 1)a - \delta^2) \mod p$ for $a$. This can be done probabilistically in polynomial time using, for example, the algorithm of Berlekamp [10]. So the case $\mathbf{a} = [a \;\; a + \delta]^T$, with known $\delta$, can be attacked using SPACDP on the system

$$m_0 + \varrho p,\; m_1 + \delta^{-1}(c_{11} - c_{12})a + r_1 p, \ldots,$$
$$m_n + \delta^{-1}(c_{n1} - c_{n2})a + r_n p.$$

□

Without knowing the parameter $\delta = a_2 - a_1$, HE2 cannot be reduced to SPACDP in this way, so HE2 is more secure than HE1.

#### *4.1.7 Cryptanalysis.*

Each new ciphertext $\mathbf{c}$ introduces two new unknowns $r, s$ and two equations for $c_1, c_2$. Thus we gain no additional information from a new ciphertext. However, if we can guess, $m$, $m'$ for any two ciphertexts $\mathbf{c}, \mathbf{c}'$, then

$$(c_1 - m) = rp + sa_1, \qquad (c_2 - m) = rp + sa_2,$$
$$(c_1' - m') = r'p + s'a_1, \qquad (c_2' - m') = r'p + s'a_2,$$

and so we have

$$(c_1 - m)(c_2' - m') - (c_2 - m)(c_1' - m') = (a_2 - a_1)(rs' - r's)p \pmod{pq}.$$

Since $a_2 \neq a_1$, and $sr' \neq s'r$ with high probability, this is a nonzero multiple of $p$, $\nu p$ say. We may assume $\nu < q$, so $p = \gcd(\nu p, pq)$. We can now solve the linear system $\gamma^T[\mathbf{c} \;\; \mathbf{c}'] = [m \;\; m'] \mod p$ to recover the decryption vector. This effectively breaks the system, since we can now decrypt an arbitrary ciphertext. We could proceed further, and attempt to infer $a_1$ and $a_2$, but we will not do so.

Note that to break this system, we need to guess two plaintexts, as opposed to one in HE1. The entropy of a pair $(m, m')$ is $2\rho$, so we have effectively squared the number of guesses needed to break the system relative to HE1. So HE2 can tolerate smaller entropy than HE1. We note further that HE2 does not seem immediately vulnerable to known cryptanalytic attacks on HE1 [50, 28, 24].

### 4.2 Insufficient entropy (HE2N)

In this section we extend HE1N above (section 3.2) to two dimensions. As with HE1N, we now encrypt by also adding a multiple of $\kappa$ to produce the ciphertext

$$\mathbf{c} = (m + rp + s\kappa)\mathbf{1} + t\mathbf{a}.$$

We decrypt by eliminating $a_1$ and $a_2$, as in HE2, but we now take residues modulo $p$ and modulo $\kappa$ to recover the plaintext,

$$m = (\boldsymbol{\gamma}^T\mathbf{c} \mod p) \mod \kappa.$$

Our addition and multiplication operations are identical to HE2 above.

#### *4.2.1 Key and Parameter Generation.*

KGen (Algorithm 17) randomly chooses $p, q$ and $\kappa$ according to the bounds given in section 3.2. $R$ is generated by Pgen (which exactly the same as Algorithm 12). The secret key is $(\kappa, p, \mathbf{a})$, and the public parameters are $pq$ and $R$, defined in section 4.1.

**Algorithm 17:** KGen: Key Generation Algorithm

**Input** : $\lambda \in \mathcal{S}$
**Input** : $\rho \in \mathbb{Z}$: entropy of input
**Input** : $\rho' \in \mathbb{Z}$: effective entropy of inputs
**Output:** $(\kappa, p, \mathbf{a})$: secret key
**Output:** modulus: public modulus
$\nu \leftarrow \rho' - \rho$
$\kappa \leftarrow_{\mathsf{p}} [2^{\nu-1}, 2^{\nu}]$
$p \leftarrow_{\mathsf{p}} [2^{\lambda-1}, 2^{\lambda}]$
$\eta \leftarrow \lambda^2/\rho' - \lambda$
$q \leftarrow_{\mathsf{p}} [2^{\eta-1}, 2^{\eta}]$
modulus $\leftarrow pq$
**repeat**
  $a_i \leftarrow_{\$} [1, \text{modulus})$ $(i \in [1, 2])$
**until** $a_1, a_2, a_1 - a_2 \neq 0$ *(*mod $p$ *and* mod $q$*)*
$\mathbf{a} \leftarrow [a_1 \;\; a_2]^T$
**return** $(\kappa, p, \mathbf{a})$, modulus

#### *4.2.2 Encryption.*

We encrypt a plaintext integer $m \in [0, M)$ as a 2-vector $\mathbf{c}$, using Enc (Algorithm 18). $\mathbf{1}$ is defined as in section 4.1.

#### *4.2.3 Decryption.*

We decrypt a ciphertext $\mathbf{c}$ using Dec (Algorithm 19).

**Algorithm 18:** Enc: Encryption Algorithm

**Input** : $m \in \mathcal{M}$
**Input** : $(\kappa, p, \mathbf{a})$: secret key
**Input** : modulus: public modulus
**Output:** $\mathbf{c} \in \mathcal{C}$
$q \leftarrow \mathsf{modulus}/p$
$r \leftarrow_\$ [0, q)$
$s \leftarrow_\$ [0, \kappa)$
$t \leftarrow_\$ [0, \mathsf{modulus})$
$\mathbf{c} \leftarrow (m + rp + s\kappa)\mathbf{1} + t\mathbf{a} \pmod{\mathsf{modulus}}$
**return** $\mathbf{c}$

**Algorithm 19:** Dec: Decryption Algorithm

**Input** : $\mathbf{c} \in \mathcal{C}$
**Input** : $(\kappa, p, \mathbf{a})$: secret key
**Output:** $m \in \mathcal{M}$
$\boldsymbol{\gamma}^T \leftarrow (a_2 - a_1)^{-1}[a_2 \ \ -a_1]$
$m \leftarrow (\boldsymbol{\gamma}^T \mathbf{c} \mod p) \mod \kappa$
**return** $m$

*4.2.4 Arithmetic.*

Addition and multiplication of ciphertexts are done using algorithms Add and Mult (which are exactly the same as Algorithms 5 and 16).

*4.2.5 Security.*

HE2N has all the properties of HE1N. However, it is more secure, since there is an additional unknown parameter in the ciphertext. We also note that HE2N satisfies Theorem 2, so it inherits the IND-CPA property.

*4.2.6 Noise Growth*

The "noise" grows as detailed in section 3.2.7.

## 5 Generalisation to $k$ dimensions

In this section, we generalise HE2 and HE2N to $k$-vectors. HE1 and HE1N are the cases for $k = 1$ and HE2 and HE2N are the cases for $k = 2$.

### 5.1 Sufficient entropy (HE$k$)

We generalise HE2 to $k$ dimensions. We extend our definition of an augmented vector $\mathbf{v}_\star$, for a $k$-vector, $\mathbf{v}$, such that $\mathbf{v}_\star$ is a $\binom{k+1}{2}$-vector, with components $v_i$ $(1 \le i \le k)$ followed by $2v_i - v_j$ $(1 \le i < j \le k)$. For example, if $k = 3$, then $\mathbf{v}_\star = [v_1, v_2, v_3, 2v_1 - v_2, 2v_1 - v_3, 2v_2 - v_3]$. In general, for $\ell > k$, $v_\ell = 2v_i - v_j$, where $\ell = \binom{i}{2} + k + j - 1$. Note that $\mathbf{v}_\star = U_k\mathbf{v}$ for a $\binom{k+1}{2} \times k$ matrix with entries $0, \pm 1, 2$, and whose first $k$ rows form the $k \times k$ identity matrix $I_k$. For example, if $k = 3$,

$$U_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix}$$

Note that $\mathbf{v}_\star = U_k\mathbf{v}$ implies that $\mathbf{1}_\star$ is the $\binom{k}{2}$ vector of 1's, and that $*$ is a linear mapping, i.e. $(r_1\mathbf{v}_1 + r_2\mathbf{v}_2)_\star = r_1\mathbf{v}_{1*} + r_2\mathbf{v}_{2*}$.

*5.1.1 Key Generation*

**Algorithm 20:** KGen: Key Generation Algorithm

**Input** : $\lambda \in \mathcal{S}$
**Input** : $\rho \in \mathbb{Z}$: entropy of inputs
**Output:** $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$: secret key
**Output:** modulus: public modulus
$p \leftarrow_\mathsf{p} [2^{\lambda-1}, 2^\lambda]$
$\eta \leftarrow \lambda^2/\rho - \lambda$
$q \leftarrow_\mathsf{p} [2^{\eta-1}, 2^\eta]$
$\mathsf{modulus} \leftarrow pq$
$\mathbf{a}_0 \leftarrow \mathbf{1}$
**repeat**
  $\mathbf{a}_j \leftarrow_\$ [1, \mathsf{modulus})^k, j \in [1, k)$
**until** $\mathbf{a}_j, j \in [0, k)$, *form a basis for* $\mathbb{Z}_{pq}^k$
**return** $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1}), \mathsf{modulus}$

KGen is detailed in Algorithm 20. It chooses $p$ and $q$ randomly, according to the bounds given in section 4.1. In addition to choosing $p$ and $q$, KGen also randomly chooses vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{k-1}$ such that $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1}$ form a basis for $\mathbb{Z}_{pq}^k$, where $\mathbf{a}_0 = \mathbf{1}$, the $k$-vector whose elements are all 1. We denote $A_k$ as the $k \times k$ matrix $[\mathbf{a}_0\ \mathbf{a}_1\ \ldots\ \mathbf{a}_{k-1}]$ where the columns of $A_k$ are the $\mathbf{a}_i, i \in [0, k)$. We show that the probability that $\mathbf{a}_1, \ldots, \mathbf{a}_{k-1}$ do not form a basis is negligible for large $p$ and $q$ in Lemma 5.

**Lemma 5** $\Pr(\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1}$ *do not form a basis for* $\mathbb{Z}_{pq}^k) \le (k-1)(1/p + 1/q)$.

*Proof* The $\mathbf{a}$'s are a basis if $A_k^{-1}$ exists, since then $\mathbf{v} = A_k\mathbf{r}$ when $\mathbf{r} = A_k^{-1}\mathbf{v}$, for any $\mathbf{v}$. Now $A_k^{-1}$ exists mod $pq$ if $(\det A_k)^{-1} \mod pq$ exists, by constructing the adjugate of $A_k$. Now $(\det A_k)^{-1} \mod pq$ exists if $\det A_k \neq 0 \mod p$ and $\det A_k \neq 0 \mod q$. Now $\det A_k$ is a polynomial of total degree $(k-1)$ in the $a_{ij}$ $(0 < i \le k, 0 < j < k)$, and is not identically zero, since $\det A_k = 1$ if $\mathbf{a}_i = \mathbf{e}_{i+1}$ $(1 < i < k)$. Also $a_{ij} \leftarrow_\$ [0, pq)$ implies

$a_{ij} \bmod p \leftarrow_\$ [0,p)$ and $a_{ij} \bmod q \leftarrow_\$ [0,q)$. Hence, using the Schwartz-Zippel Lemma (SZL) [61], we have $\Pr(\det A_k = 0 \bmod p) \leq (k-1)/p$ and $\Pr(\det A_k = 0 \bmod q) \leq (k-1)/q$, and it follows that

$\Pr(\nexists\, (\det A_k)^{-1} \bmod pq) \;\leq\; (k-1)(1/p + 1/q).$ □

*5.1.2 Parameter Generation*

**Algorithm 21:** Pgen: Parameter Generation Algorithm

**Input :** $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$: secret key
**Input :** modulus: public modulus
**Output:** $R$: public re-encryption matrix

$q \leftarrow \mathsf{modulus}/p$
$\mathbf{a}_0 \leftarrow \mathbf{1}$
$A_k \leftarrow [\mathbf{a}_0\ \mathbf{a}_1\ \ldots\ \mathbf{a}_{k-1}]$
**foreach** $j \in [0,k)$ **do**
  $\mathbf{a}_{\star j} \leftarrow U_k \mathbf{a}_j$
**end**
**foreach** $i,j \in [0,k), 0 \leq i \leq j < k$ **do**
  $\mathbf{a}^{\circ 2}_{\star,ij} \leftarrow \mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}$
**end**
$A^{\circ 2}_{\star k} \leftarrow [\mathbf{a}^{\circ 2}_{\star,01}\ \ldots\ \mathbf{a}^{\circ 2}_{\star,ij}\ \ldots\ \mathbf{a}^{\circ 2}_{\star,k-2,k-1}]$ // $A^{\circ 2}_{\star k}$ is a $\binom{k+1}{2} \times \binom{k+1}{2}$ matrix
**if** $A^{\circ 2}_{\star k}$ *has no inverse* **then**
  **return** $\perp$
**end**
**foreach** $i,j \in [0,k), 0 \leq i \leq j < k$ **do**
  $\varrho_{ij} \leftarrow_\$ [0,q)$
  **foreach** $l \in [1,k)$ **do**
    $\sigma_{ijl} \leftarrow_\$ [0, \mathsf{modulus})$
  **end**
  $\mathbf{b}_{ij} \leftarrow \varrho_{ij} p\mathbf{1} + \sum_{l=1}^{k-1} \sigma_{ijl} \mathbf{a}_l$
**end**
$C_k \leftarrow [\mathbf{b}_{01}\ \ldots\ \mathbf{b}_{ij}\ \ldots\ \mathbf{b}_{k-2,k-1}]$ // $C_k$ is a $k \times \binom{k}{2}$ matrix
$D_k \leftarrow [A_k \mid C_k]$ // $D_k$ is a $k \times \binom{k+1}{2}$ matrix
$R \leftarrow D_k (A^{\circ 2}_{\star k})^{-1}$ // $R$ is a $k \times \binom{k+1}{2}$ matrix
**return** $R$

Pgen generates the $k \times \binom{k+1}{2}$ re-encryption matrix, $R$, as detailed in Algorithm 21. We construct $R$ so that it satisfies equation 3

$$R(\mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}) \;=\; \varrho_{ij} p\mathbf{1} + \textstyle\sum_{l=1}^{k-1} \sigma_{ijl} \mathbf{a}_l \qquad (3)$$

This ensures that, if $\mathbf{c}$ is an encryption of $m$ and $\mathbf{c}'$ is an encryption of $m'$, then $R(\mathbf{c}_\star \circ \mathbf{c}'_\star) \pmod{pq}$ is a valid encryption of $mm'$.

We also construct $R$ so that it satisfies the following identity:

**Lemma 6** *Let $A_{\star k} = [\mathbf{a}_{\star 0}\ \mathbf{a}_{\star 1}\ \ldots\ \mathbf{a}_{\star,k-1}]$, where the columns of $A_k$ form a basis for $\mathbb{Z}^k_{pq}$. If $RA_{\star k} = A_k$, then $R\mathbf{v}_\star = \mathbf{v}$ for all $\mathbf{v} \in \mathbb{Z}^k_{pq}$.*

*Proof* We have $\mathbf{v} = A_k \mathbf{r}$ for some $\mathbf{r} \in \mathbb{Z}^k_{pq}$. Then $A_{\star k} = U_k A_k$ and $\mathbf{v}_{\star k} = U_k \mathbf{v}$, so $R\mathbf{v}_\star = RU_k\mathbf{v} = RU_kA_k\mathbf{r} = RA_{\star k}\mathbf{r} = A_k\mathbf{r} = \mathbf{v}$. □

This guarantees that re-encrypting a ciphertext by applying $R$ reveals no new information.

Pgen returns a valid matrix $R$ provided the $A^{\circ 2}_{\star k}$ has an inverse. We prove that this is true with high probability in Theorem 5. In the unlikely event that this is not true, Pgen exits and we use KGen to generate new vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{k-1}$ until it is.

**Theorem 5** *$A^{\circ 2}_{\star k}$ has no inverse* mod$pq$ *with probability at most $(k^2-1)(1/p + 1/q)$.*

*Proof* We use the same approach as in Lemma 5. Thus $A^{\circ 2}_{\star k}$ is invertible provided $\det A^{\circ 2}_{\star k} \neq 0 \pmod p$ and $\det A^{\circ 2}_{\star k} \neq 0 \pmod q$. Let $\mathbf{A}$ denote the vector of $a_{ij}$'s, $(a_{ij} : 1 \leq i \leq k, 1 \leq j < k)$. The elements of $A^{\circ 2}_{\star k}$ are quadratic polynomials over $\mathbf{A}$, except for the first column, which has all 1's, and columns $2, 3, \ldots, k$ which are linear polynomials. So $\det A^{\circ 2}_{\star k}$ is a polynomial over $\mathbf{A}$ of total degree $2\binom{k}{2} + k - 1 = k^2 - 1$. Thus, unless $\det A^{\circ 2}_{\star k}$ is identically zero as a polynomial over $\mathbf{A}$, the SZL [61] implies $\Pr(\nexists\, (\det A^{\circ 2}_{\star k})^{-1} \bmod p) \leq (k^2-1)/p$ and $\Pr(\nexists\, (\det A^{\circ 2}_{\star k})^{-1} \bmod q) \leq (k^2-1)/q$. Therefore we have $\Pr(\nexists\, (\det A^{\circ 2}_{\star k})^{-1} \bmod pq) \leq (k^2-1)(1/p + 1/q)$.

It remains to prove that $\det A^{\circ 2}_{\star k}$ is not identically zero as a polynomial over $\mathbf{A}$ in either $\mathbb{Z}_p$ or $\mathbb{Z}_q$. We prove this by induction on $k$. Consider $\mathbb{Z}_p$, the argument for $\mathbb{Z}_q$ being identical. Since $\mathbb{Z}_p$ is a field, $\det A^{\circ 2}_{\star k}$ is identically zero if and only if it has rank less than $\binom{k+1}{2}$ for all $\mathbf{A}$. That is, there exist $\lambda_{ij}(\mathbf{A}) \in \mathbb{Z}_p$ $(0 \leq i \leq j < k)$, not all zero, so that

$$\begin{aligned}\mathcal{L}(\mathbf{A}) \;&=\; \sum_{0 \leq i \leq j}^{k-1} \lambda_{ij} \mathbf{a}_{\star i} \circ \mathbf{a}_{\star j} \\ &= \boldsymbol{\alpha} + \mathbf{a}_{\star,k-1} \circ \boldsymbol{\beta} + \lambda_{k-1,k-1} \mathbf{a}^{\circ 2}_{\star,k-1} \;=\; 0,\end{aligned}$$

where $\boldsymbol{\alpha} = \sum_{0 \leq i \leq j}^{k-2} \lambda_{ij} \mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}$ and $\boldsymbol{\beta} = \sum_{i=0}^{k-2} \lambda_{i,k-1} \mathbf{a}_{\star i}$ are independent of $\mathbf{a}_{\star,k-1}$.

Clearly $\lambda_{k-1,k-1} = 0$. Otherwise, whatever $\boldsymbol{\alpha}, \boldsymbol{\beta}$, we can choose values for $\mathbf{a}_k$ so that $\mathcal{L} \neq 0$, a contradiction. Now suppose $\lambda_{i,k-1} \neq 0$ for some $0 \leq i < k-1$. The matrix $\hat{A}_\star$ with columns $\mathbf{a}_{\star i}$ $(0 \leq i < k-1)$ contains $A_{k-1}$ as a submatrix, which has rank $(k-1)$ with high probability by Lemma 5. Thus $\beta \neq \mathbf{0}$ and, whatever $\boldsymbol{\alpha}$, we can choose values for $\mathbf{a}_k$ so that $\mathcal{L} \neq 0$. Thus $\lambda_{i,k-1} = 0$ for all $0 \leq i < k$. Thus $\lambda_{ij} \neq 0$ for some $0 \leq i \leq j < k-1$. Now the matrix $\hat{A}^{\circ 2}_\star$ with $\binom{k}{2}$ columns $\mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}$ $(0 \leq i \leq j < k-1)$ contains $A^{\circ 2}_{\star,k-1}$ as a submatrix, and therefore has rank $\binom{k}{2}$ by induction. Hence $\alpha \neq \mathbf{0}$, implying $\mathcal{L} \neq 0$, a contradiction. □

Note that Theorem 5 subsumes Lemma 5, since the first $k$ columns of $A^{\circ 2}_{\star k}$ contain $A_k$ as a submatrix, and must be linearly independent.

*5.1.3 Computational overhead*

The computational overhead increases, since the number of arithmetic operations per plaintext multiplication is $O(k^3)$, and the space requirement per ciphertext is $O(k)$, by comparison with HE1.

*5.1.4 Encryption*

**Algorithm 22:** Enc: Encryption Algorithm

**Input** : $m \in \mathcal{M}$
**Input** : $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$: secret key
**Input** : modulus: public modulus
**Output:** $\mathbf{c} \in \mathcal{C}$

$q \leftarrow \mathsf{modulus}/p$
$r \leftarrow_{\$} [0, q)$
**foreach** $j \in [1, k)$ **do**
  $s_j \leftarrow_{\$} [0, \mathsf{modulus})$
**end**
$\mathbf{c} \leftarrow (m + rp)\mathbf{1} + \sum_{j=1}^{k-1} s_j \mathbf{a}_j \pmod{\mathsf{modulus}}$
**return** $\mathbf{c}$

A plaintext, $m \in [0, M]$, is enciphered as $\mathbf{c}$, a $k$-vector, using Algorithm 22.

*5.1.5 Decryption.*

**Algorithm 23:** Dec: Decryption Algorithm

**Input** : $\mathbf{c} \in \mathcal{C}$
**Input** : $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$): secret key
**Output:** $m \in \mathcal{M}$

$A_k \leftarrow [\mathbf{a}_0\ \mathbf{a}_1\ \ldots\ \mathbf{a}_{k-1}]$
$\boldsymbol{\gamma}^T \leftarrow (A_k^{-1})_1$ // $(A_k^{-1})_1$ is the first row of $A_k^{-1}$
$m \leftarrow \boldsymbol{\gamma}^T \mathbf{c} \mod p$
**return** $m$

A ciphertext is decrypted using Algorithm 23. We call $\boldsymbol{\gamma}$ the *decryption vector*, as in HE2.

*5.1.6 Addition.*

Addition is the vector sum of the ciphertext vectors as in HE2 (see section 4.1). The Add algorithm is identical to Algorithm 15 with the exception that the vectors involved are $k$-vectors rather than 2-vectors.

*5.1.7 Multiplication*

Consider a Hadamard product of two augmented ciphertext vectors, $\mathbf{c}_\star \circ \mathbf{c}'_\star$. For notational brevity, let $\tilde{m} = m + rp$.

$$\begin{aligned}\mathbf{c}_\star \circ \mathbf{c}'_\star &= \left(\tilde{m}\mathbf{1}_\star + \textstyle\sum_{j=1}^{k-1} s_j \mathbf{a}_{\star j}\right) \circ \left(\tilde{m}'\mathbf{1}_\star + \textstyle\sum_{j=1}^{k-1} s'_j \mathbf{a}_{\star j}\right) \\ &= \tilde{m}\tilde{m}'\mathbf{1}_\star + \textstyle\sum_{j=1}^{k-1}(\tilde{m}s'_j + \tilde{m}'s_j)\mathbf{a}_{\star j} \\ &\quad + \textstyle\sum_{j=1}^{k-1} s_j s'_j \mathbf{a}_{\star j} \circ \mathbf{a}_{\star j} \\ &\quad + \textstyle\sum_{1 \le i < j \le k-1}(s_i s'_j + s'_i s_j)\mathbf{a}_{\star i} \circ \mathbf{a}_{\star j},\end{aligned}$$

since $\mathbf{1}_\star \circ \mathbf{v}_\star = \mathbf{v}_\star$ for any $\mathbf{v}$. There are $\binom{k}{2}$ product vectors, which we must eliminate using the re-encryption matrix $R$, a $k \times \binom{k+1}{2}$ matrix.

From Lemma 6, we have that $R\mathbf{v}_{\star k} = \mathbf{v}$ for all $\mathbf{v} \in \mathbb{Z}_{pq}^k$. Therefore, $RA_{\star k} = A_k$. However, this condition can be written more simply, since it is $RU_kA_k = A_k$. Postmultiplying by $A_k^{-1}$ gives $RU_k = I_k$. For example, if $k = 3$,

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} & R_{15} & R_{16} \\ R_{21} & R_{22} & R_{23} & R_{24} & R_{25} & R_{26} \\ R_{31} & R_{32} & R_{33} & R_{34} & R_{35} & R_{36} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

So $R_{11} + 2R_{14} + 2R_{15} = 1$, $R_{11} = 1 - 2R_{14} - 2R_{15}$, etc. Now, since $RA_{\star k} = A_k$, we have

$$\begin{aligned}R(\mathbf{c}_\star \circ \mathbf{c}'_\star) &= (mm' + \hat{r}p)\mathbf{1} + \textstyle\sum_{j=1}^{k-1} \hat{s}_j \mathbf{a}_j \\ &\quad + \textstyle\sum_{1 \le i \le j \le k-1} \hat{s}_{ij} R(\mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}),\end{aligned}$$

where $\hat{r}$, $\hat{s}_j$ and $\hat{s}_{ij}$ $(1 \le i < j \le k-1)$ are some integers.

There are $k\binom{k}{2}$ undetermined parameters $R_{i\ell}$, $1 \le i \le k$, $k < \ell \le \binom{k+1}{2}$. We now determine these by setting

$$R(\mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}) = \varrho_{ij} p\mathbf{1} + \textstyle\sum_{l=1}^{k-1} \sigma_{ijl}\, \mathbf{a}_l$$

Thus we have $k\binom{k}{2}$ new unknowns, the $\varrho$'s and $\sigma$'s, and $k\binom{k}{2}$ linear equations for the $k\binom{k}{2}$ unassigned $R_{i\ell}$'s. Let $A^{\circ 2}_{\star k}$ be the $\binom{k+1}{2} \times \binom{k+1}{2}$ matrix with columns $\mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}$ $(0 \le i < j < k)$, and let $C_k$ be the $k \times \binom{k}{2}$ matrix with columns $\varrho_{ij} p\mathbf{1} + \sum_{l=1}^{k-1} \sigma_{ijl}\, \mathbf{a}_l$ $(0 < i < j < k)$. Then the equations for the $R_{i\ell}$ can be written as

$$RA^{\circ 2}_{\star k} = [A_k \mid C_k].$$

which by construction of $R$ (Algorithm 21) has a solution.

The multiplication algorithm Mult is given by Algorithm 24.

**Algorithm 24:** Mult: multiplication algorithm

**Input** : $\mathbf{c} = [c_1\ c_2]^T \in \mathcal{C}$
**Input** : $\mathbf{c}' = [c'_1\ c'_2]^T \in \mathcal{C}$
**Input** : modulus $\in \mathbb{Z}$: public modulus
**Input** : $R$: re-encryption matrix
**Output:** result $\in \mathcal{C}$

$\mathbf{c}_\star \leftarrow U_k\mathbf{c}$
$\mathbf{c}'_\star \leftarrow U_k\mathbf{c}'$
result $\leftarrow R(\mathbf{c}_\star \circ \mathbf{c}'_\star)$ (mod modulus)
**return** result

*5.1.8 Security*

We construct $R$ so that it satisfies Equation 4:

$$RA_{\star k}^{\circ 2} \;=\; [A_k \mid C_k]\,. \tag{4}$$

which gives $k\binom{k+1}{2}$ linear equations for the $k\binom{k+1}{2}$ elements of $R$ in terms of quadratic functions of the $k(k-1)$ $a_{ij}$'s $(1 \le i \le k, 1 \le j \le k-1)$, which are undetermined. Thus the system has $k(k-1)$ parameters that cannot be deduced from $R$.

Each $\mathbf{c}$ introduces $k$ new parameters $rp, s_1, \ldots, s_{k-1}$ and $k$ equations, so the number of undetermined parameters is always $k(k-1)$.

*5.1.9 Cryptanalysis*

Note that $p$ can be determined from $m_i$ for $k$ ciphertexts. Let

$$C = [\mathbf{c}_1 - m_1\mathbf{1}\ \ldots\ \mathbf{c}_k - m_k\mathbf{1}],$$
$$A_k = [\mathbf{1}\ \mathbf{a}_1\ \ldots\ \mathbf{a}_{k-1}],$$

and let

$$W = \begin{bmatrix} r_1p & r_2p & \ldots & r_kp \\ s_{1,1} & s_{2,1} & \ldots & s_{k,1} \\ \vdots & & & \vdots \\ s_{1,k-1} & s_{2,k-1} & \ldots & s_{k,k-1} \end{bmatrix},$$

$$W' = \begin{bmatrix} r_1 & r_2 & \ldots & r_k \\ s_{1,1} & s_{2,1} & \ldots & s_{k,1} \\ \vdots & & & \vdots \\ s_{1,k-1} & s_{2,k-1} & \ldots & s_{k,k-1} \end{bmatrix},$$

where $r_i, s_{ij}$ refer to $\mathbf{c}_i$. Then $C = A_kW$, and so $\det C = \det A_k \det W$. Note that $\det W = p \det W'$, so $\det C$ is a multiple of $p$. Now $\det C$ can be determined in $O(k^3)$ time and, if it is nonzero, $p$ can be determined as $\gcd(\det C, pq)$.

**Lemma 7** $\Pr(\det C = 0 \bmod pq) \le (2k-1)(1/p + 1/q)$.

*Proof* From Lemma 5, $\det A = 0 \mod p$ or $\det A = 0 \mod q$ with probability at most $(k-1)(1/p + 1/q)$. So $\det A$ is not zero or a divisor of zero $\bmod pq$. The entries of $W'$ are random $[0, pq)$, and $\det W'$ is a polynomial of total degree $k$ in its entries. It is a nonzero polynomial, since $W' = I_k$ is possible. Hence, using the SZL [61], $\Pr(\det W' = 0 \bmod p) \le k/p$ and $\Pr(\det W' = 0 \bmod q) \le k/q$. So $\det W'$ is zero or a divisor of zero $\bmod pq$ with probability at most $k(1/p + 1/q)$. So we have $\det A \det W' = 0 \bmod pq$ with probability at most $(2k-1)(1/p + 1/q)$. So $\det C \ne 0$ with high probability. □

Once we have recovered $p$, we can use the known $m_i$ to determine the decryption vector $\boldsymbol{\gamma}$, by solving a set of linear equations. Let $C_0 = [\mathbf{c}_1\ \mathbf{c}_2\ \ldots\ \mathbf{c}_k]$, and $\mathbf{m}^T = [m_1\ m_2\ \ldots\ m_k]$.

**Lemma 8** $\Pr(\det C_0 = 0 \bmod pq) \le (2k-1)(1/p + 1/q)$.

*Proof* Note that $C_0 = C$ if $m_1 = m_2 = \cdots = m_k = 0$. Since Lemma 7 holds in that case, the result follows. □

Thus, with high probability, we can uniquely solve the system $\boldsymbol{\gamma}^T C_0 = \mathbf{m}^T \bmod p$, to recover $\boldsymbol{\gamma}$ and enable decryption of an arbitrary ciphertext. However, encryption of messages is not possible, since we gain little information about $\mathbf{a}_1, \ldots, \mathbf{a}_k$. Note also that, if we determined $p$ by some means other than using $k$ known plaintexts, it is not clear how to recover $\boldsymbol{\gamma}$.

To break this system, we need to guess $k$ plaintexts. The entropy of a $k$-tuple of plaintexts $(m_1, m_2, \ldots, m_k)$ is $k\rho$, so effectively we need $\mu^k$ guesses, where $\mu$ is the number of guesses needed to break HE1. So HE$k$ can tolerate much smaller entropy than HE1, provided $k$ is large enough. If $k$ is sufficiently large, the scheme appears secure without adding noise, but does not have the other advantages of adding noise.

*5.1.10 Fixing an insecurity for $k > 2$*

The decryption vector for HE$k$ is $\boldsymbol{\gamma}^T = (A_k^{-1})_1$. Note that $\boldsymbol{\gamma}^T\mathbf{1} = 1$ and $\boldsymbol{\gamma}^T\mathbf{a}_i = 0$ $(i \in [1, k-1])$, since $\boldsymbol{\gamma}^T\mathbf{a}_i = I_{1i}$ $(i \in [0, k-1])$. The equations

$$R(\mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}) \;=\; p\varrho_{ij}\,\mathbf{1} + \textstyle\sum_{l=1}^{k-1} \sigma_{ijl}\,\mathbf{a}_l, \tag{5}$$

define a product $\cdot$ on $\mathbb{Z}_{pq}^k$ so that $\mathbf{c}\cdot\mathbf{c}' = R(\mathbf{c}_\star \circ \mathbf{c}'_\star)$. This product is linear, commutative and distributive, since $R$ and $\star$ are linear operators, and $\circ$ is commutative and distributive. So we have an algebra $\mathcal{A}_k$, with unit element $\mathbf{1}$ [72]. The $\varrho_{ij}, \sigma_{ijl}$ $(i, j, l \in [1, k-1])$ are the *structure constants* of the algebra. In general, $\mathcal{A}_k$ will

**Algorithm 25:** Pgen: Parameter Generation Algorithm (Revised)

**Input** : $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$: secret key
**Input** : modulus: public modulus
**Output:** $R$: public re-encryption matrix

$q \leftarrow \text{modulus}/p$
$\mathbf{a}_0 \leftarrow \mathbf{1}$
$A_k \leftarrow [\mathbf{a}_0\ \mathbf{a}_1\ \ldots\ \mathbf{a}_{k-1}]$
**foreach** $j \in [0, k)$ **do**
  $\mathbf{a}_{\star j} \leftarrow U_k \mathbf{a}_j$
**end**
**foreach** $i, j \in [0, k), 0 \le i \le j < k$ **do**
  $\mathbf{a}^{\circ 2}_{\star,ij} \leftarrow \mathbf{a}_{\star i} \circ \mathbf{a}_{\star j}$
**end**
$A^{\circ 2}_{\star k} \leftarrow [\mathbf{a}^{\circ 2}_{\star,01}\ \ldots\ \mathbf{a}^{\circ 2}_{\star,ij}\ \ldots\ \mathbf{a}^{\circ 2}_{\star,k-2,k-1}]$ // $A^{\circ 2}_{\star k}$ is a $\binom{k+1}{2} \times \binom{k+1}{2}$ matrix
**if** $A^{\circ 2}_{\star k}$ *has no inverse* **then**
  **return** $\perp$
**end**
**foreach** $i, j \in [0, k), 0 \le i \le j < k$ **do**
  $\varrho_i \leftarrow_\$ [0, q)$
  $\varrho_i \leftarrow_\$ [0, q)$
  $\varrho_{ij} \leftarrow \varrho_i \varrho_j \mod q$
  **repeat**
    $\tau \leftarrow_\$ [0, q)$
    **foreach** $l \in [1, k)$ **do**
      $\sigma'_{ijl} \leftarrow_\$ [0, q)$
    **end**
  **until** $\sum_{l=1}^{k-1} \sigma'_{ijl} \varrho_i = \tau \varrho_i \varrho_j \mod q$
  **foreach** $l \in [1, k)$ **do**
    $r_{ijl} \leftarrow_\$ [0, p)$
    $\sigma_{ijl} \leftarrow \sigma'_{ijl} + r_{ijl} q$
  **end**
  $\mathbf{b}_{ij} \leftarrow \varrho_{ij} p \mathbf{1} + \sum_{l=1}^{k-1} \sigma_{ijl} \mathbf{a}_l$
**end**
$C_k \leftarrow [\mathbf{b}_{01}\ \ldots\ \mathbf{b}_{ij}\ \ldots\ \mathbf{b}_{k-2,k-1}]$ // $C_k$ is a $k \times \binom{k}{2}$ matrix
$D_k \leftarrow [A_k \mid C_k]$ // $D_k$ is a $k \times \binom{k+1}{2}$ matrix
$R \leftarrow D_k (A^{\circ 2}_{\star k})^{-1}$ // $R$ is a $k \times \binom{k+1}{2}$ matrix
**return** $R$

not be associative, i.e. we can have $(\mathbf{c}_1 \cdot \mathbf{c}_2) \cdot \mathbf{c}_3 \neq \mathbf{c}_1 \cdot (\mathbf{c}_2 \cdot \mathbf{c}_3)$ This leads to a potential insecurity. We must have

$$\boldsymbol{\gamma}^T((\mathbf{c}_1 \cdot \mathbf{c}_2) \cdot \mathbf{c}_3) \;=\; \boldsymbol{\gamma}^T(\mathbf{c}_1 \cdot (\mathbf{c}_2 \cdot \mathbf{c}_3)) \qquad (\text{mod } p), \quad (6)$$

in order to have correct decryption. The *associator* for $\mathcal{A}_k$ is

$$\begin{aligned}[\mathbf{c}_i, \mathbf{c}_j, \mathbf{c}_l] &= \mathbf{c}_i \cdot (\mathbf{c}_j \cdot \mathbf{c}_l) - (\mathbf{c}_i \cdot \mathbf{c}_j) \cdot \mathbf{c}_l \\ &= rp\mathbf{1} + \textstyle\sum_{l=1}^{k-1} s_l \mathbf{c}_l \qquad (\text{mod } pq).\end{aligned}$$

Thus $[\mathbf{c}_i, \mathbf{c}_j, \mathbf{c}_l]$ is an encryption of 0. If we can find $k$ associators from $\mathbf{c}_1, \ldots, \mathbf{c}_n$ which violate (6), with high probability we have $k$ linearly independent associators. We can use these to make a collision attack on HE$k$, similar to that described in section 3.1. We use the gcd method to determine $p$, and then $\boldsymbol{\gamma}$, as described in section 5.1.9. In fact all we need is that (6) holds for any associator. That is, for all $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$, we need

$$\boldsymbol{\gamma}^T((\mathbf{c}_1 \cdot \mathbf{c}_2) \cdot \mathbf{c}_3) \;=\; \boldsymbol{\gamma}^T(\mathbf{c}_1(\cdot \mathbf{c}_2 \cdot \mathbf{c}_3)) \qquad (\text{mod } pq),$$

or, equivalently, using the Chinese Remainder Theorem,

$$\boldsymbol{\gamma}^T((\mathbf{c}_1 \cdot \mathbf{c}_2) \cdot \mathbf{c}_3) \;=\; \boldsymbol{\gamma}^T(\mathbf{c}_1 \cdot (\mathbf{c}_2 \cdot \mathbf{c}_3)) \qquad (\text{mod } q). \quad (7)$$

By linearity, (7) holds if and only if it holds for all basis elements, excluding the identity. That is, for all $i, j, l \in [1, k-1]$, we need

$$\boldsymbol{\gamma}^T(\mathbf{a}_i \cdot (\mathbf{a}_j \cdot \mathbf{a}_l)) \;=\; \boldsymbol{\gamma}^T((\mathbf{a}_i \cdot \mathbf{a}_j) \cdot \mathbf{a}_l) \qquad (\text{mod } q). \quad (8)$$

The associator for $\mathcal{A}_k$ is

$$\begin{aligned}[\mathbf{a}_i, \mathbf{a}_j, \mathbf{a}_l] &= \mathbf{a}_i \cdot (\mathbf{a}_j \cdot \mathbf{a}_l) - (\mathbf{a}_i \cdot \mathbf{a}_j) \cdot \mathbf{a}_l \\ &= rp\mathbf{1} + \textstyle\sum_{l=1}^{k-1} s_l \mathbf{a}_l \qquad (\text{mod } pq),\end{aligned}$$

for some integers $r, s_1, \ldots, s_{k-1}$, and so $\boldsymbol{\gamma}^T[\mathbf{a}_i, \mathbf{a}_j, \mathbf{a}_l] = rp$.

If $\mathcal{A}_k$ is associative, the problem does not arise, since (8) will be satisfied automatically. Associativity holds if $k \le 2$. All we have to check is that $\mathbf{a} \cdot (\mathbf{a} \cdot \mathbf{a}) = (\mathbf{a} \cdot \mathbf{a}) \cdot \mathbf{a}$, which is true by commutativity. Thus HE1, HE2 cannot be attacked in this way. However, this collision attack becomes possible even when $k = 3$.

*Example 1* Suppose that $\mathbf{a}_1 \cdot \mathbf{a}_1 = \mathbf{a}_2 \cdot \mathbf{a}_2 = p\mathbf{1} + \mathbf{a_1} - \mathbf{a_2}$, $\mathbf{a}_1 \cdot \mathbf{a}_2 = 2p\mathbf{1} + \mathbf{a_2}$. Then $\mathbf{a}_1 \cdot (\mathbf{a}_1 \cdot \mathbf{a}_2) = \mathbf{a}_1 \cdot (2p\mathbf{1} + \mathbf{a}_2) = 2p\mathbf{a_1} + \mathbf{a_1} \cdot \mathbf{a_2} = 2p\mathbf{1} + 2p\mathbf{a_1} + \mathbf{a}_2$, and $(\mathbf{a}_1 \cdot \mathbf{a}_1) \cdot \mathbf{a}_2 = p\mathbf{a}_2 + \mathbf{a_1} \cdot \mathbf{a_2} + \mathbf{a_2} \cdot \mathbf{a_2} = 3p\mathbf{1} + \mathbf{a}_1 + p\mathbf{a}_2$. So $[\mathbf{a}_1, \mathbf{a}_1, \mathbf{a}_2] = p\mathbf{1} - (2p-1)\mathbf{a}_1 + (p+1)\mathbf{a}_2$, and $\boldsymbol{\gamma}^T[\mathbf{a}_1, \mathbf{a}_1, \mathbf{a}_2] = p$.

Requiring associativity in $\mathcal{A}_k$ overconstrains the system, imposing $k\binom{k+1}{2}$ equations on the $k\binom{k+1}{2}$ structure constants. With only $k(k-1)$ undetermined parameters, this is too much. But all we need is that (8) holds. We have

**Lemma 9** *Equation* (8) *holds if and only if, for all* $i, j, l \in [1, k-1]$, $\sum_{t=1}^{k-1} \sigma_{jlt} \varrho_{it} = \sum_{t=1}^{k-1} \sigma_{ijt} \varrho_{lt} \pmod q$.

*Proof* Since $\boldsymbol{\gamma}^T \mathbf{1} = 1$ and $\boldsymbol{\gamma}^T \mathbf{a}_i = 0$, $i \in [1, k-1]$, $\boldsymbol{\gamma}^T(\mathbf{a}_i \cdot \mathbf{a}_j) = \boldsymbol{\gamma}^T\big(p\varrho_{ij}\,\mathbf{1} + \sum_{l=1}^{k-1} \sigma_{ijl} \mathbf{a}_l\big) = p\varrho_{ij}$. Thus

$$\begin{aligned}\mathbf{a}_i \cdot (\mathbf{a}_j \cdot \mathbf{a}_l) &= \mathbf{a}_i \cdot \big(p\varrho_{jl}\mathbf{1} + \textstyle\sum_{t=1}^{k-1} \sigma_{jlt} \mathbf{a}_t\big) \\ &= p\varrho_{jl}\mathbf{a}_i + \textstyle\sum_{t=1}^{k-1} \sigma_{jlt} \mathbf{a}_i \cdot \mathbf{a}_t,\end{aligned}$$

and hence $\boldsymbol{\gamma}^T[\mathbf{a}_i \cdot (\mathbf{a}_j \cdot \mathbf{a}_l)] = p\sum_{t=1}^{k-1} \sigma_{jlt} \varrho_{it}$. Similarly $\boldsymbol{\gamma}^T[(\mathbf{a}_i \cdot \mathbf{a}_j) \cdot \mathbf{a}_l] = p\sum_{t=1}^{k-1} \sigma_{ijt} \varrho_{lt}$, and the lemma follows. □

Now we can ensure (8) by giving the $\varrho_{ij}$ a multiplicative structure.

**Lemma 10** *Let $\tau, \varrho_i \leftarrow_\$ [0,q)$ $(i \in [1,k-1])$, let $\varrho_{ij} = \varrho_i\varrho_j \mod q$, and let the $\sigma_{ijl}$ satisfy $\sum_{l=1}^{k-1} \sigma_{ijl}\varrho_l = \tau\varrho_i\varrho_j \pmod q$ for all $i,j \in [1,k-1]$. Then, for all $i,j,\ell \in [1,k-1]$, $\boldsymbol{\gamma}^T(\mathbf{a}_i \cdot (\mathbf{a}_j \cdot \mathbf{a}_l)) = \tau\varrho_i\varrho_j\varrho_l \mod q$, the symmetry of which implies* (8).

*Proof* We have $\boldsymbol{\gamma}^T(\mathbf{a}_j \cdot \mathbf{a}_l) = p\varrho_{ij} = p\varrho_j\varrho_l$ for all $j,\ell \in [1,k-1]$. Hence, mod $q$,

$$\begin{aligned}\boldsymbol{\gamma}^T(\mathbf{a}_i \cdot (\mathbf{a}_j \cdot \mathbf{a}_l)) &= p\textstyle\sum_{t=1}^{k-1}\sigma_{jlt}\varrho_{it}\\ &= p\textstyle\sum_{t=1}^{k-1}\sigma_{jlt}\varrho_i\varrho_t\\ &= p\varrho_i\textstyle\sum_{t=1}^{k-1}\sigma_{jlt}\varrho_t\\ &= p\varrho_i\tau\varrho_j\varrho_l \;=\; p\tau\varrho_i\varrho_j\varrho_l.\end{aligned}$$

Thus the conditions of Lemma 10 are sufficient to remove the insecurity. The price is that we now have $(k-1)\binom{k}{2}+(k-1)+k(k-1) = (k+1)\binom{k}{2}+k-1$ parameters and $k\binom{k}{2}$ equations. There are $\binom{k}{2}+(k-1) = (k+2)(k-1)/2$ independent parameters. This is fewer than the original $k(k-1)$, but remains $\Omega(k^2)$.

As a result, the parameter generation algorithm Pgen is amended to Algorithm 25.

## 5.2 Insufficient entropy (HE$k$N)

We generalise HE2N to $k$ dimensions.

### *5.2.1 Key generation*

**Algorithm 26:** KGen: Key Generation Algorithm

**Input** : $\lambda \in \mathcal{S}$
**Input** : $\rho \in \mathbb{Z}$: entropy of inputs
**Input** : $\rho' \in \mathbb{Z}$: entropy of inputs
**Output:** $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$: secret key
**Output:** modulus: public modulus

$\nu \leftarrow \rho' - \rho$
$\kappa \leftarrow_\mathsf{p} [2^{\nu-1}, 2^\nu]$
$p \leftarrow_\mathsf{p} [2^{\lambda-1}, 2^\lambda]$
$\eta \leftarrow \lambda^2/\rho - \lambda$
$q \leftarrow_\mathsf{p} [2^{\eta-1}, 2^\eta]$
modulus $\leftarrow pq$
$\mathbf{a}_0 \leftarrow \mathbf{1}$
**repeat**
  $\mathbf{a}_j \leftarrow_\$ [1, \text{modulus})^k, j \in [1,k)$
**until** $\mathbf{a}_j, j \in [0,k)$, *form a basis for* $\mathbb{Z}_{pq}^k$
**return** $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$, modulus

KGen (Algorithm 26, randomly chooses $\kappa$, $p$ and $q$ according to the bounds outlined in section 4.2, and sets $\mathbf{a}_j$ $\forall j$. Note that, as a result of adding the "noise" term, defence against non-associativity is not required. Therefore, Pgen, which generates $R$, is Algorithm 21. The secret key, sk, is $(\kappa, p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$, and the public parameters are $pq$ and $R$.

### *5.2.2 Encryption*

**Algorithm 27:** Enc: Encryption Algorithm

**Input** : $m \in \mathcal{M}$
**Input** : $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$: secret key
**Input** : modulus: public modulus
**Output:** $\mathbf{c} \in \mathcal{C}$

$q \leftarrow \text{modulus}/p$
$r \leftarrow_\$ [0,q)$
$s \leftarrow_\$ [0,\kappa)$
**foreach** $j \in [1,k)$ **do**
  $t_j \leftarrow_\$ [0, \text{modulus})$
**end**
$\mathbf{c} \leftarrow (m + rp + s\kappa)\mathbf{1} + \sum_{j=1}^{k-1} t_j\mathbf{a}_j \pmod{\text{modulus}}$
**return** $\mathbf{c}$

A plaintext, $m \in [0,M]$, is enciphered by Algorithm 27.

### *5.2.3 Decryption*

**Algorithm 28:** Dec: Decryption Algorithm

**Input** : $\mathbf{c} \in \mathcal{C}$
**Input** : $(p, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1})$): secret key
**Output:** $m \in \mathcal{M}$

$A_k \leftarrow [\mathbf{a}_0\ \mathbf{a}_1\ \ldots\ \mathbf{a}_{k-1}]$
$\boldsymbol{\gamma}^T \leftarrow (A_k^{-1})_1$ `// (A_k^{-1})_1 is the first row of A_k^{-1}`
$m \leftarrow (\boldsymbol{\gamma}^T\mathbf{c} \mod p) \mod \kappa$
**return** $m$

A ciphertext is deciphered by Algorithm 28.

### *5.2.4 Arithmetic*

The addition and multiplication algorithms are as in section 5.1.

### *5.2.5 Security*

The effective entropy of HE$k$N is $\rho' = k(\rho + \lg\kappa)$. Thus, as we increase $k$, the "noise" term can be made smaller while still providing the requisite level of entropy.

Clearly HE$k$N also inherits the conclusions of Theorem 2, so this system also satisfies IND-CPA.

### *5.2.6 Noise Growth*

The "noise" grows as detailed in section 3.2.7.

## 6 An extension of HE2N using the Chinese Remainder Theorem (HE2NCRT)

As an interesting aside, we extend HE2N (section 4.2) using a technique inspired by Chinese Remainder Theorem (CRT) secret sharing, so that we compute the final result modulo a product of primes $\prod_{j=1}^{K} p_j$ rather than modulo $p$, where $K$ is the number of primes.

In this scheme, we distribute the computation. We have $K$ sets of processors. Each processor computes arithmetic on ciphertexts modulo $p_j q_j$, where $p_j, q_j$ are suitable primes. Each plaintext is encrypted as $K$ ciphertexts where each processor receives the $j$th ciphertext vector. Addition and multiplication of ciphertexts on processor $j$ is as defined in section 4.1, except that it is performed modulo $p_j q_j$.

This serves two purposes. The first is to be able to handle larger moduli by dividing the computation into subcomputations on smaller moduli. The second is to mitigate against exposure of the secret key $p$ in the system presented in section 4.2, by not distributing the modulus $pq$ to each processor. Instead, we distribute $p_j q_j$ to the $j$th processor, for $j \in [1, K]$. This allows us to replicate the computation where each subcomputation is encrypted using different parameters. Thus, should an attacker compromise one subcomputation, they may gain no knowledge of other subcomputations.

### *6.0.1 Key Generation*

**Algorithm 29:** KGen: Key Generation Algorithm

**Input :** $\lambda \in \mathcal{S}$
**Input :** $\rho \in \mathbb{Z}$: entropy of input
**Input :** $\rho' \in \mathbb{Z}$: effective entropy of inputs
**Output:** $(\kappa, p_1, p_2, \ldots, p_K, \mathbf{a}_1, \ldots, \mathbf{a}_K)$: secret key
**Output:** $\mathsf{modulus}_1, \mathsf{modulus}_2, \ldots, \mathsf{modulus}_K$: public moduli

$\nu \leftarrow \rho' - \rho$
$\kappa \leftarrow_{\mathsf{p}} [2^{\nu-1}, 2^{\nu}]$
$\eta \leftarrow \lambda^2/\rho' - \lambda$
**foreach** $j \in [1, K]$ **do**
  $p_j \leftarrow_{\mathsf{p}} [2^{\lambda-1}, 2^{\lambda}]$
  $q_j \leftarrow_{\mathsf{p}} [2^{\eta-1}, 2^{\eta}]$
  $\mathsf{modulus}_j \leftarrow p_j q_j$
  **repeat**
    $a_j i \leftarrow_{\$} [1, \mathsf{modulus}_j)$ $(i \in [1, 2])$
  **until** $a_{j1}, a_{j2}, a_{j1} - a_{j2} \neq 0$ (mod $p$ *and* mod $q$)
  $\mathbf{a}_j \leftarrow [a_{j1}\ a_{j2}]^T$
**end**
**return** $(\kappa, p_1, p_2, \ldots, p_K, \mathbf{a}_1, \ldots, \mathbf{a}_K)$, $\mathsf{modulus}_1$, $\mathsf{modulus}_2, \ldots, \mathsf{modulus}_K$

The key generation process, KGen (Algorithm 29) randomly chooses $\kappa$ as in section 3.2. For all $j \in [1, K]$, it randomly chooses a prime $p_j$ such that $p_j$ satisfies $2^{\lambda-1} < p_j < 2^{\lambda}$ and

$$\Pi = \prod_{j=1}^{K} p_j > (n+1)^d (M + \kappa^2)^d.$$

It also randomly chooses $q_j$, $j \in [1, K]$, as for $q$ in section 3.1. Finally, it sets $\mathbf{a}_j = [a_{j1}\ a_{j2}]^T$, where $a_{jk} \leftarrow_{\$} [1, p_j q_j)$ $(j \in [1, K], k \in [1, 2])$ such that $a_{j1} \neq a_{j2} \pmod{p}$ and $a_{j1} \neq a_{j2} \pmod{q}$.

The parameter generation routine Pgen (Algorithm 30) generates each re-encrytion matrix $R_j$ $(j \in [1, K])$.

The secret key, $\mathbf{sk}$, is $(\kappa, p_1, \ldots, p_K, \mathbf{a}_1, \ldots, \mathbf{a}_K)$, and the public parameters are $p_j q_j$ $(j \in [1, K])$ and $R_j$ $(j \in [1, K])$.

**Algorithm 30:** Pgen: Parameter Generation Algorithm

**Input :** $(\kappa, p_1, p_2, \ldots, p_K, \mathbf{a}_1, \ldots, \mathbf{a}_K)$: secret key
**Input :** $\mathsf{modulus}_1, \mathsf{modulus}_2, \ldots, \mathsf{modulus}_K$: public moduli
**Output:** $R_1, R_2, \ldots, R_K$: public re-encryption matrices

**foreach** $j \in [1, K]$ **do**
  $q_j \leftarrow \mathsf{modulus}_j / p_j$
  $\beta_j \leftarrow 2(a_{j2} - a_{j1})^2$
  $\varrho_j \leftarrow_{\$} [0, q_j)$
  $\sigma_j \leftarrow_{\$} [0, \mathsf{modulus}_j)$
  $\alpha_{j1} \leftarrow \beta_j^{-1}(\sigma_j a_{j1} + \varrho_j p - a_{j1}^2)$
  $\alpha_{j2} \leftarrow \beta_j^{-1}(\sigma_j a_{j2} + \varrho_j p - a_{j2}^2)$
  $R_j \leftarrow \begin{bmatrix} 1 - 2\alpha_{j1} & \alpha_{j1} & \alpha_{j1} \\ -2\alpha_{j2} & \alpha_{j2} + 1 & \alpha_{j2} \end{bmatrix}$
**end**
**return** $R_1, \ldots, R_K$

### *6.0.2 Encryption*

**Algorithm 31:** Enc: Encryption Algorithm

**Input :** $m \in \mathcal{M}$
**Input :** $(\kappa, p_1, p_2, \ldots, p_K, \mathbf{a}_1, \ldots, \mathbf{a}_K)$: secret key
**Input :** $\mathsf{modulus}_1, \mathsf{modulus}_2, \ldots, \mathsf{modulus}_K$: public moduli
**Output:** $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K \in \mathcal{C}$

**foreach** $j \in [1, K]$ **do**
  $q_j \leftarrow \mathsf{modulus}_j / p_j$
  $r_j \leftarrow_{\$} [0, q_j)$
  $s_j \leftarrow_{\$} [0, \kappa)$
  $t_j \leftarrow_{\$} [0, \mathsf{modulus}_j)$
  $\mathbf{c}_j \leftarrow (m + r_j p_j + s_j \kappa)\mathbf{1} + t_j \mathbf{a}_j \pmod{\mathsf{modulus}_j}$
**end**
**return** $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K$

We encrypt an integer, $m \in \mathcal{M}$, as the set of $K$ 2-vectors, $\mathbf{c}_j$ using Algorithm 31.

*6.0.3 Decryption*

**Algorithm 32:** Dec: Decryption Algorithm

**Input** : $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K \in \mathcal{C}$
**Input** : $(\kappa, p_1, p_2, \ldots, p_K, \mathbf{a}_1, \ldots, \mathbf{a}_K)$: secret key
**Output:** $m \in \mathcal{M}$

$\Pi \leftarrow \prod_{j=1}^{K} p_j$
**foreach** $j \in [1, K]$ **do**
  $\boldsymbol{\gamma}_j^T \leftarrow (a_{j2} - a_{j1})^{-1}[a_{j2} \quad -a_{j1}]$
  $d_j \leftarrow (\boldsymbol{\gamma}_j^T \mathbf{c}_j \mod p_j)$
  $M_j \leftarrow \Pi / p_j$
  $\mu_j \leftarrow M_j^{-1} \pmod{p_j}$
**end**
$m \leftarrow \left( \sum_{j=1}^{K} d_j M_j \mu_j \pmod{\Pi} \right) \pmod{\kappa}$
**return** $m$

To decrypt a set of $K$ ciphertexts we use Algorithm 32. We first decrypt the $j$th ciphertext of the computational result, $\mathbf{c}_j$, as in section 4.1, to give

$$P_j = (a_{j2} - a_{j1})^{-1}(a_{j2}c_{j1} - a_{j1}c_{j2}) \pmod{p_j},$$

where $P_j = P(m_1, m_2, \ldots, m_n, \kappa) \pmod{p_j}$.

We then use the Chinese Remainder Theorem to compute the plaintext as

$$P(m_1, m_2, \ldots, m_n) = \left( \sum_{j=1}^{K} P_j M_j \mu_j \pmod{\Pi} \right) \pmod{\kappa},$$

where $M_j = \Pi / p_j$ and $\mu_j = M_j^{-1} \pmod{p_j}$. Note that $\mu_j$ is a residue rather than a plaintext value.

Note that learning $p_j$ on one processor, does not allow an attacker to reconstruct the $m_i$. Rather they only learn $m_i + s_{ij}\kappa \pmod{p}_j$.

*6.0.4 Arithmetic*

**Algorithm 33:** $\mathsf{Add}_j$: addition algorithm for processor $j$

**Input** : $\mathbf{c}_j \in \mathcal{C}$
**Input** : $\mathbf{c}'_j \in \mathcal{C}$
**Input** : $\mathsf{modulus}_j \in \mathbb{Z}$: modulus for arithmetic
**Output:** $\mathsf{result}_j \in \mathcal{C}$

$\mathsf{result}_j \leftarrow \mathbf{c}_j + \mathbf{c}'_j \pmod{\mathsf{modulus}_j}$
**return** $\mathsf{result}_j$

Addition of ciphertexts on processor $j$ is performed using Algorithm 33. Multiplication of ciphertexts on processor $j$ is performed by Algorithm 34.

**Algorithm 34:** $\mathsf{Mult}_j$: multiplication algorithm for processor $j$

**Input** : $\mathbf{c}_j = [c_{j1} \; c_{j2}]^T \in \mathcal{C}$
**Input** : $\mathbf{c}'_j = [c'_{j1} \; c'_{j2}]^T \in \mathcal{C}$
**Input** : $\mathsf{modulus}_j \in \mathbb{Z}$: public modulus
**Input** : $R_j$: re-encryption matrix
**Output:** $\mathsf{result}_j \in \mathcal{C}$

$c_{j3} \leftarrow 2c_{j1} - c_{j2}$
$\mathbf{c}_{j\star} \leftarrow [c_{j1} \; c_{j2} \; c_{j3}]^T$
$c'_{j3} \leftarrow 2c'_{j1} - c'_{j2}$
$\mathbf{c}'_{j\star} \leftarrow [c'_{j1} \; c'_{j2} \; c'_{j3}]^T$
$\mathsf{result}_j \leftarrow R_j(\mathbf{c}_{j\star} \circ \mathbf{c}'_{j\star}) \pmod{\mathsf{modulus}_j}$
**return** $\mathsf{result}_j$

*6.0.5 Extending HE2NCRT to k-vectors*

Clearly HE$k$N could be extended to HE$k$NCRT in a similar way to the extension of HE2 to HE$k$. Future work will discuss the details of such a scheme.

## 7 Fully Homomorphic System

We return to HE$k$, presented above in section 5.1. We will show that, for large enough $k$, this can be made into a fully homomorphic system.

Suppose we have inputs $m_1, m_2, \ldots, m_n$, where $m_i \in [0, M]$, $i \in [1, n]$, $p$, $q$ are large primes, and we want to evaluate an arithmetic circuit, $\Phi$, over the ring $\mathbb{Z}_{pq}$, on these inputs.

From [51], the definition of an arithmetic circuit is:

**Definition 4 (Arithmetic Circuits.)** An arithmetic circuit $\Phi$ over the ring $\mathsf{R}$ and variables $X = \{x_1, \ldots, x_n\}$ is a directed acyclic graph with every node of in-degree either two or zero, labelled in the following manner: every vertex of in-degree 0 is labelled by either a variable in $X$ or an element of $\mathsf{R}$. Every other node in $\Phi$ has in-degree two and is labelled by either $\times$ or $+$. A circuit $\Phi$ computes a polynomial $f \in \mathsf{R}[X]$ in the obvious manner. An arithmetic circuit is called a formula if the out-degree of each node in it is one (and so the underlying graph is a directed tree). The size of a circuit is the number of nodes in it, and the depth of a circuit is the length of the longest directed path in it.

To transform HE$k$ to a fully homomorphic system, we define encryption of the circuit inputs as in HE$k$. Similarly, addition and multiplication of ciphertexts at each arithmetic node of the circuit is defined as in the HE$k$ scheme. This way we are able to compute the arithmetic circuit homomorphically. However, this system is still "somewhat" homomorphic. If the computational result grows larger than $p$, we are unable to

successfully decrypt the result. This restricts us to circuits of bounded depth to avoid this plaintext blow up. To make it fully homomorphic, we consider Boolean circuits [82]. A Boolean circuit is defined in [82] as:

**Definition 5 (Boolean circuit.)** We define a basis of Boolean functions, $B$, where each member of $B$ is a function, $f : [0,1]^m \to [0,1]$, for some $m \in \mathbb{N}$. Therefore, a Boolean circuit over a basis $B$, with $n$ inputs and $m$ outputs, is then defined as a finite directed acyclic graph. Each vertex corresponds to either a basis function, which we call a *gate*, or one of the inputs, and there are a set of exactly $m$ nodes which are labeled as the outputs. The edges must also have some ordering, to distinguish between different arguments to the same Boolean function.

We note that any finite computation can be represented as a Boolean circuit.

Typically, the basis $B$ will be the Boolean functions, AND, OR, and NOT. However, a Boolean circuit may be alternatively represented using only NAND gates [73]. The indegree of any gate in the directed acyclic graph $G = (V, E)$ is then always 2, but the outdegree may be arbitrary. We will refer to the directed edges of this graph as *wires*. We note that, for a Boolean circuit, the inputs to each gate are bits, as are the outputs. We will denote the set of inputs to the circuit by $I \subseteq V$, and the set of outputs from the circuit by $O \subseteq V$. In $G$, the inputs have indegree 0, and the outputs have outdegree 0, but we will regard the inputs as having indegree 1, and the outputs as having outdegree 1, with wires from and to the external environment $\Lambda$.

Note that, if we represent the bit values $0, 1$ with known constants $\alpha_0, \alpha_1$, the system is open to attack, even though the inputs are encrypted with HE$k$. For any ciphertext $\mathbf{c}$, we can take

$$\mathbf{c}' = \begin{cases} \mathbf{c} - \alpha_0\mathbf{1}, \text{ with probability } \frac{1}{2}, \\ \mathbf{c} - \alpha_1\mathbf{1}, \text{ with probability } \frac{1}{2}. \end{cases}$$

Then $\mathbf{c}'$ is an encryption of 0 with probability $\frac{1}{2}$. If we multiply $\nu$ of these ciphertexts, the probability that none of them is an encryption of zero is $2^{-\nu}$, so we obtain an encryption of zero with probability $1 - 2^{-\nu}$. So, with $\nu$ repetitions of this process, if $\nu = 2 \lg k$, this is $1 - 1/k$. By repeating the whole process $\Omega(k)$ times, we can obtain $k$ encryptions of zero with high probability. Once we have done this, we can use the gcd method of 5.1.9 to determine $p$.

Therefore, we encrypt both the inputs to the circuit, and the values on the wires, using HE$k$. On each wire $e \in E$, we will represent the bit value $b_e \in \{0, 1\}$ by $w_e \in \{\omega_{0e}, \omega_{1e}\}$, where $\omega_{0e}, \omega_{1e} \in [0, q)$ are random even and odd integers respectively. Thus $b_e = w_e \bmod 2$. For each input $i \in I$, we represent the input bit value $b_i$ similarly, by $x_i \in \{\omega_{0i}, \omega_{1i}\}$.

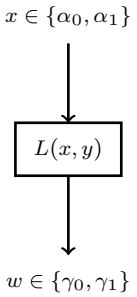


**Figure 1** A wire from an input

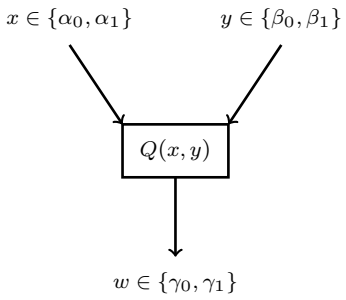


**Figure 2** An output wire on a NAND gate

An input $i \in I$ has a wire $(\Lambda, i)$ on which the (encrypted) input value $x_i$ is stored. For any wire $e = (i, v)$ from input $i$, we have a linear function $L(x) = a + bx$, which converts the plaintext input value $x \in \{\alpha_0, \alpha_1\}$ to the wire value $w \in \{\gamma_0, \gamma_1\}$. It is easy to check that this requires

$$a = (\alpha_1 - \alpha_0)^{-1}(\alpha_1\gamma_0 - \alpha_0\gamma_1),$$
$$b = (\alpha_1 - \alpha_0)^{-1}(\gamma_1 - \gamma_0).$$

The encrypted coefficients of this function are stored as data for the wire $e$ (see Fig. 1). Note that all computations are mod $pq$, and the required inverses exist because the numbers involved are less than $q$.

For each output wire $e = (v, v')$ of a NAND gate $v$, we have a quadratic function $Q(x, y) = a + bx + cy + dxy$, which converts the values on the input wires of the gate, $x \in \{\alpha_0, \alpha_1\}$, $y \in \{\beta_0, \beta_1\}$, to the wire value $w \in \{\gamma_0, \gamma_1\}$. It is easy to check that this requires

$$a = \gamma_0 + \alpha_1\beta_1\vartheta, \quad b = -\beta_1\vartheta, \quad c = -\alpha_1\vartheta, \quad d = \vartheta,$$

where $\vartheta = \big((\alpha_1 - \alpha_0)(\beta_1 - \beta_0)\big)^{-1}(\gamma_1 - \gamma_0)$. Again, the encrypted coefficients of this function are stored as data for the wire $e$ (see Fig. 2).

For each output NAND gate $v \in O$, we decrypt the value $w \in \{\gamma_0, \gamma_1\}$ computed by its (unique) output wire $(v, \Lambda)$. Then the output bit is $w \bmod 2$.

Thus we have replaced the logical operations of the Boolean circuit by evaluation of low degree polynomials. For simplicity, we have chosen to use only NAND

gates, but we can represent any binary Boolean function by a quadratic polynomial in the way described above. Since the quadratic polynomials are encrypted in our system, they conceal the binary Boolean function they represent. Thus the circuit can be "garbled" [8,45], to minimise inference about the inputs and outputs of the circuit from its structure.

However, there is a price to be paid for controlling the noise. The encrypted circuit is not securely reusable with the same values $\omega_{0e}, \omega_{1e}$ for $w_e$. Suppose we can observe the encrypted value on wire $e$ three times giving cyphertexts $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$. Two of these are encryptions of the same value $2s_{0,e}$ or $1 + 2s_{1,e}$. Thus $(\mathbf{c}_1 - \mathbf{c}_2) \cdot (\mathbf{c}_1 - \mathbf{c}_3) \cdot (\mathbf{c}_2 - \mathbf{c}_3)$ is an encryption of 0. By doing this for $k$ wires, we can break the system. This is essentially the collision attack described in section 3.

Some reuse of the encrypted circuit is possible by using multiple values on the wires, and higher degree polynomials for the gates. However, we will not consider this refinement, since the idea seems to have little practical interest.

As we discuss in section 5.1, the security of HE$k$ is exponential in $k$, So by setting $k$ large enough we make brute force attacks on the system infeasible. (A brute force attack succeeds in polynomial time only if $k = O(\log \log p)$.) Therefore, if we compute encrypted Boolean circuits, our system is fully homomorphic.

## 8 Experimental Results

HE1, HE1N, HE2, and HE2N have been implemented in pure unoptimised Java using the JScience mathematics library [34]. Secure pseudo-random numbers are generated using the ISAAC algorithm [52], seeded using the Linux `/dev/random` source. This prevents the weakness in ISAAC shown by Aumasson [5].

The evaluation experiment generated 24,000 encrypted inputs and evaluated a polynomial homomorphically on the inputs, using a Hadoop MapReduce (MR) algorithm. On the secure client side, the MR input is generated as pseudo-random $\rho$-bit integers which are encrypted and written to a file with $d$ inputs per line, where $d$ is the degree of the polynomial to be computed. The security parameters $\lambda$ and $\eta$ were selected to be the minimum values required to satisfy the conditions give in sections 3.1, 3.2, 4.1, and 4.2. In addition, the unencrypted result of the computation is computed so that it may checked against the decrypted result of the homomorphic computation. On the Hadoop cluster side, each mapper processes a line of input by homomorphically multiplying together each input on a line and outputs this product. A single reducer homomorphically sums the products. The MR algorithm divides the input file so that each mapper receives an equal number of lines of input, ensuring maximum parallelisation. Finally, on the secure client side, the MR output is decrypted.

Our test environment consisted of a single secure client (an Ubuntu Linux VM with 16GB RAM) and a Hadoop 2.7.3 cluster running in a heterogeneous OpenNebula cloud. The Hadoop cluster consisted of 17 Linux VMs, one master and 16 slaves, each allocated 2GB of RAM. Each experimental configuration of algorithm, polynomial degree ($d$), integer size ($\rho$), and effective entropy of inputs after adding "noise" ($\rho'$, for the 'N' variant algorithms only), was executed 10 times. The means are tabulated in Table 1.

There are some small anomalies in our data. JScience implements arbitrary precision integers as an array of Java `long` (64-bit) integers. This underlying representation may be optimal in some of our test configurations and suboptimal in others, causing anomalous results. Another possibility is that the unexpected results are due to garbage collection in the JVM heap, which may be more prevalent in certain test configurations.

We may compare these results with those reported in the literature. Our results compare extremely favourably with Table 2 of [57]. For encryption, our results are, in the best case, 1000 times faster than those presented there, and, in the worst case, 10 times faster. For decryption, our results are comparable. However, it should be noted that to decrypt our results we take the moduli for large primes rather than 2 as in [57], which is obviously less efficient. For homomorphic sums and products, our algorithms perform approximately 100 times faster. [57] only provides experimental data for computing degree 2 polynomials. We provide experimental results for higher degree polynomials.

Similarly, compared with Fig. 13 of Popa et al. [65], our encryption times for a 32-bit integer are considerably faster. While a time for computing a homomorphic sum on a column is given in Fig. 12, it is unclear how many rows exist in their test database. Nevertheless, our results for computing homomorphic sums compare favourably with those given. Since CryptDB [65] only supports homomorphic sums and cannot compute an inner product, we can only compare the homomorphic sum timings.

Table 1 of [76] is unclear whether the timings are aggregate or per operation. Even assuming that they are aggregate, our results are approximately 100 times faster for homomorphic sum and product operations. Crypsis [76] uses two different encryption schemes for integers, ElGamal [40] and Paillier [63], which only support addition or multiplication but not both. No discussion of computation of an inner product is made in [76]

**Table 1** Timings for each experimental configuration ($n = 24000$ in all cases, $\lambda > 96$). *Init* is the initialisation time for the encryption algorithm, *Enc* is the mean time to encrypt a single integer, *Exec* is the total MR job execution time, *Prod* is the mean time to homomorphically compute the product of two encrypted integers, *Sum* is the mean time to homomorphically compute the sum of two encrypted integers.

| Alg. | Parameters | | | Encryption | | MR Job | | | Decrypt |
|---|---|---|---|---|---|---|---|---|---|
| | $d$ | $\rho$ | $\rho'$ | Init(s) | Enc($\mu$s) | Exec(s) | Prod($\mu$s) | Sum($\mu$s) | (ms) |
| HE1 | 2 | 32 | n/a | 0.12 | 13.52 | 23.82 | 54.41 | 9.06 | 0.21 |
| HE1 | 2 | 64 | n/a | 0.12 | 16.24 | 23.85 | 60.38 | 8.04 | 0.49 |
| HE1 | 2 | 128 | n/a | 0.15 | 25.73 | 23.77 | 84.69 | 8.43 | 0.28 |
| HE1 | 3 | 32 | n/a | 0.17 | 22.98 | 23.65 | 87.75 | 11.46 | 0.35 |
| HE1 | 3 | 64 | n/a | 0.19 | 34.63 | 24.72 | 95.68 | 12.37 | 0.45 |
| HE1 | 3 | 128 | n/a | 0.42 | 54.83 | 26.05 | 196.71 | 14.07 | 0.55 |
| HE1 | 4 | 32 | n/a | 0.28 | 43.36 | 24.48 | 108.72 | 13.75 | 0.5 |
| HE1 | 4 | 64 | n/a | 0.53 | 58.85 | 26.41 | 227.44 | 15.85 | 3.59 |
| HE1 | 4 | 128 | n/a | 1.36 | 104.95 | 28.33 | 484.95 | 16.92 | 5.67 |
| HE1N | 2 | 1 | 32 | 0.22 | 32.99 | 22.94 | 88.38 | 8.53 | 3.35 |
| HE1N | 2 | 1 | 64 | 0.39 | 52.63 | 26.24 | 168.54 | 12.39 | 3.56 |
| HE1N | 2 | 1 | 128 | 1.2 | 89.01 | 26.18 | 226.2 | 13.16 | 8.1 |
| HE1N | 2 | 8 | 32 | 0.6 | 57.88 | 25.9 | 177.36 | 11.17 | 7.18 |
| HE1N | 2 | 8 | 64 | 0.32 | 43.93 | 26.53 | 96.78 | 12.18 | 2.27 |
| HE1N | 2 | 8 | 128 | 1.13 | 78.11 | 24.42 | 212.75 | 11.07 | 8.4 |
| HE1N | 2 | 16 | 64 | 0.33 | 53.97 | 27.15 | 168 | 13.67 | 4.47 |
| HE1N | 2 | 16 | 128 | 0.63 | 68.73 | 25.22 | 194.42 | 11.01 | 7.65 |
| HE1N | 3 | 1 | 32 | 8.54 | 183.19 | 24.24 | 522.07 | 12.06 | 9.09 |
| HE1N | 3 | 1 | 64 | 3.67 | 125 | 29.49 | 467.36 | 18.22 | 11.43 |
| HE1N | 3 | 1 | 128 | 27.84 | 313.76 | 26.94 | 1235.77 | 15.04 | 11.75 |
| HE1N | 3 | 8 | 32 | 115 | 462.45 | 32.61 | 1556.17 | 21.11 | 19.79 |
| HE1N | 3 | 8 | 64 | 9.75 | 180.08 | 25.87 | 500.62 | 15.03 | 10.39 |
| HE1N | 3 | 8 | 128 | 36.05 | 259.15 | 30.1 | 836.27 | 20.68 | 11.45 |
| HE1N | 3 | 16 | 64 | 30.96 | 378.99 | 28.24 | 1338.33 | 15.51 | 13.3 |
| HE1N | 3 | 16 | 128 | 8.13 | 226.32 | 27.92 | 621.95 | 18.01 | 10.89 |
| HE2 | 2 | 32 | n/a | 0.16 | 85.79 | 26.82 | 305.52 | 11.68 | 4.83 |
| HE2 | 2 | 64 | n/a | 0.17 | 95.92 | 29.71 | 354.79 | 16.9 | 3.26 |
| HE2 | 2 | 128 | n/a | 0.22 | 132.53 | 32.84 | 540.78 | 22.83 | 4.92 |
| HE2 | 3 | 32 | n/a | 0.23 | 130.3 | 31.18 | 513.93 | 23.77 | 6.52 |
| HE2 | 3 | 64 | n/a | 0.29 | 145.62 | 32.84 | 615.9 | 24.61 | 6.3 |
| HE2 | 3 | 128 | n/a | 0.52 | 249.47 | 29.54 | 1443.82 | 16.56 | 18.34 |
| HE2 | 4 | 32 | n/a | 0.39 | 175.63 | 29.5 | 733.23 | 20.69 | 6.01 |
| HE2 | 4 | 64 | n/a | 0.7 | 255.3 | 29.55 | 1578.39 | 18.29 | 16.24 |
| HE2 | 4 | 128 | n/a | 2.7 | 465.51 | 37.47 | 2943.91 | 22.15 | 15.41 |
| HE2N | 2 | 1 | 32 | 0.27 | 147.83 | 29.74 | 571.94 | 16.58 | 5.66 |
| HE2N | 2 | 1 | 64 | 0.43 | 202.74 | 33.36 | 1291.68 | 18.3 | 13.23 |
| HE2N | 2 | 1 | 128 | 1.58 | 354.19 | 33.76 | 1977.51 | 17.13 | 12.46 |
| HE2N | 2 | 8 | 32 | 0.59 | 234.83 | 31.42 | 1413.31 | 15.21 | 14.92 |
| HE2N | 2 | 8 | 64 | 0.33 | 163.78 | 27.42 | 635.64 | 13.6 | 6.18 |
| HE2N | 2 | 8 | 128 | 0.9 | 307.68 | 36.32 | 1850.83 | 21.71 | 15.79 |
| HE2N | 2 | 16 | 64 | 0.42 | 208.1 | 29.96 | 1230.56 | 13.41 | 13.16 |
| HE2N | 2 | 16 | 128 | 0.73 | 274.48 | 30.82 | 1585.1 | 14.85 | 15.04 |
| HE2N | 3 | 1 | 32 | 5.72 | 651.1 | 36.49 | 3438.96 | 18.67 | 19.05 |
| HE2N | 3 | 1 | 64 | 4.45 | 477.52 | 35.33 | 3073.46 | 18.75 | 19.77 |
| HE2N | 3 | 1 | 128 | 26.83 | 1192.79 | 43.23 | 6416.43 | 22.48 | 25.12 |
| HE2N | 3 | 8 | 32 | 87.38 | 1658.36 | 49.63 | 8139.19 | 23.71 | 27.24 |
| HE2N | 3 | 8 | 64 | 5.21 | 607.75 | 36.54 | 3337.1 | 22.28 | 17.39 |
| HE2N | 3 | 8 | 128 | 17.14 | 945.64 | 40.49 | 4620.69 | 25.91 | 22.41 |
| HE2N | 3 | 16 | 64 | 39.19 | 1368.18 | 44.88 | 7005.7 | 24.1 | 28.3 |
| HE2N | 3 | 16 | 128 | 11.39 | 774.07 | 36.05 | 3845.1 | 20.29 | 20.74 |

but we expect that the timings would be considerably worse as data encrypted using ElGamal to compute the products would have to be shipped back to the secure client to be re-encrypted using Paillier so that the final inner product could be computed.

Varia et al. [79] present experimental results of applying their HETest framework to HELib [48]. Varia et al. show timings $10^4$ to $10^6$ times slower than that of computations on unencrypted data. Although it is unclear exactly which circuits are being computed, the timings given are in seconds, so we believe that HELib will not be a serious candidate for SSCC in the immediate future.

As reported in [37], the current computational performance of FHEW [38] is poor compared with unencrypted operations. The authors report that FHEW processed a single homomorphic NAND operation followed by a re-encryption in 0.69s and using 2.2GB of RAM. Therefore, we also believe that FHEW is not a candidate for SCCC, as it currently stands.

Although claims regarding its performance have appeared in the press [78], no benchmarking statistics have been made publicly available for Microsoft's SEAL library [56]. However, in [3], it is reported that, for SEAL v1, the time to perform one multiplication is approximately 140ms.

With regard to FV-NFLib [33], Bonte et al. [15] recently reported a significant decrease in the time to evaluate a four layer *Group Method of Data Handling* (GMDH) neural network [16] from 32s to 2.5s, as a result of their novel encoding of the inputs.

Aguilar-Melchor et al [3] report experimental findings regarding HELib-MP [69]. They show that HELib-MP outperforms FV-NFLib for large (2048-bit) plaintexts. They further go on to benchmark HELib-MP by computing RSA-2048 and ECC-ElGamal-P256. An exponentiation in RSA-2048 takes between 157ms and 1.8s depending on the window size and number of multiplications required. For ECC-ElGamal-P256, an elliptic curve multiplication takes between 96ms and 242ms depending on window size and number of elliptic curve additions.

Catalano et al. [22] provide experimental results for their work. For 128-bit plaintexts, our algorithms are approximately 10 to 1000 times faster at performing a multiplication operation and our most complex algorithm, HE2N, is roughly equal to their fastest, an extension of Joye-Libert [53], for additions.

Yu et al. [84] give experimental results for their implementation of the Zhou and Wornell scheme [85]. From their Figures 3 to 5, it is hard to compare our scheme with theirs directly but it would appear that our vector based schemes are at least comparable in performance to theirs.

### 8.1 Microsoft Azure Cloud

In addition to performing our experiments in a small private cloud, we also scaled our experiment to a large cluster in Microsoft's Azure public cloud [4]. Our experimental environment consisted of a HDInsight cluster comprising two D13v2 head nodes and 123 D4v2 worker nodes (984 worker cores).

The number of inputs was increased to 106,272,000 for each experimental configuration. As a result of the large volume of input data, our experiments were altered to create and encrypt the data *in situ*. This encrypted data is then consumed by our experimental program. As before, the output from out experiment was downloaded to a secure client and decrypted to verify the result. In addition, as a result of the larger number of inputs, we employed a tighter upper bound ($nM^d/d$) on the values of $p$ and $\kappa$ required to successfully decrypt the HE1/HE2 and HE1N/HE2N variants respectively.

The results for each experimental configuration are presented in Table 2. Comparing it with Table 1, shows only an increase of approximately 300% in average encryption and product calculation times for our most complex algorithm (HE2N), despite the 4,428-fold increase in the number of inputs. The fact that these times did not scale linearly with the number of inputs may largely be attributed to the increased memory per worker node (28GB versus 2GB) and the tighter bounds on $p$ and $\kappa$. However, we also note that the bit size of the arbitrary precision integers involved will scale logarithmically in the number of inputs as a result of the tighter bound. Hence, the timings for arithmetic operations on those integers will also scale logarithmically.

## 9 Conclusion

In this paper we have presented several new homomorphic encryption schemes intended for use in *secure single-party computation in the cloud* (SSCC). We envisage that the majority of computation on integer big data, outside of scientific computing, will be computing low degree polynomials on integers, or fixed-point decimals which can be converted to integers. Our somewhat homomorphic schemes are perfectly suited to these types of computation.

Our evaluations have concerned only one- or two-dimensional ciphertexts and polynomials of degree up to four. We intend to investigate higher degree polynomials in future work. We believe that HE1N and HE2N provide strong security, even for low-entropy data, as they satisfy the desirable IND-CPA property. If a user has a high confidence in the entropy of the input data, HE2 may provide sufficient security.

As they are only somewhat homomorphic, each of these schemes require that the computational result cannot grow bigger than the secret modulus. In the case of the "noise" variants, we also have to consider the noise term growing large. So, as they stand, these schemes can only compute polynomials of a suitably bounded degree. However, we believe this is adequate for most practical purposes.

A further concern is that the ciphertext space is much larger than the plaintext space. This is as a result of adding multiples of large primes to the plaintext. However, we have shown that values exist which makes the system practical for computing low degree polynomials. Similar schemes [31,35] produce ciphertexts infeasibly larger than the corresponding plaintext,

[4] This work was aided by a Microsoft Azure for Research sponsorship.[60]

**Table 2** Timings for each experimental configuration ($n = 106272000$ in all cases, $\lambda > 96$). *Init* is the initialisation time for the encryption algorithm, *Enc* is the mean time to encrypt a single integer, *Exec* is the total MR job execution time, *Prod* is the mean time to homomorphically compute the product of two encrypted integers, *Sum* is the mean time to homomorphically compute the sum of two encrypted integers.

| Alg. | Parameters | | | Encryption | | MR Job | | | Decrypt |
|---|---|---|---|---|---|---|---|---|---|
| | $d$ | $\rho$ | $\rho'$ | Init(ms) | Enc($\mu$s) | Exec(s) | Prod($\mu$s) | Sum($\mu$s) | (ms) |
| HE1 | 2 | 32 | - | 85.53 | 32.28 | 28.29 | 11.47 | 2.73 | 245 |
| HE1 | 2 | 64 | - | 94.9 | 39.92 | 27.81 | 11.45 | 2.51 | 647 |
| HE1 | 2 | 128 | - | 108.41 | 54.96 | 32.37 | 29.65 | 2.42 | 315 |
| HE1 | 3 | 32 | - | 103.05 | 47.72 | 28.39 | 17.19 | 2.55 | 380 |
| HE1 | 3 | 64 | - | 133.23 | 65.26 | 35.3 | 41.01 | 2.62 | 347 |
| HE1 | 3 | 128 | - | 146.93 | 105.08 | 66.31 | 283.02 | 1.76 | 555 |
| HE1 | 4 | 32 | - | 110.88 | 62.83 | 34.23 | 41.86 | 2.54 | 561 |
| HE1 | 4 | 64 | - | 157.18 | 101.26 | 70.7 | 298.73 | 1.96 | 601 |
| HE1 | 4 | 128 | - | 2244.7 | 201.59 | 86.06 | 398.64 | 2.77 | 27720 |
| HE1N | 2 | 1 | 32 | 120.36 | 81.56 | 33.04 | 31.19 | 2.46 | 4757 |
| HE1N | 2 | 1 | 64 | 254.62 | 113.8 | 60.43 | 226.07 | 1.71 | 6093 |
| HE1N | 2 | 1 | 128 | 543.44 | 214.76 | 68.83 | 289.12 | 2.25 | 34792 |
| HE1N | 2 | 8 | 32 | 220.88 | 121.43 | 62.32 | 234.7 | 2.01 | 5854 |
| HE1N | 2 | 8 | 64 | 197.35 | 96.82 | 35.99 | 40.38 | 2.2 | 5371 |
| HE1N | 2 | 8 | 128 | 558.05 | 190.3 | 71.05 | 287.43 | 2.25 | 35071 |
| HE1N | 2 | 16 | 64 | 159.33 | 112.72 | 35.5 | 42.87 | 2.59 | 5697 |
| HE1N | 2 | 16 | 128 | 433.71 | 171.56 | 64.5 | 261.45 | 2.23 | 33943 |
| HE1N | 3 | 1 | 32 | 617.06 | 130.88 | 68.5 | 300.76 | 1.91 | 32622 |
| HE1N | 3 | 1 | 64 | 1656.53 | 265 | 284 | 1520.65 | 3.08 | 41454 |
| HE1N | 3 | 1 | 128 | 43002.78 | 709.31 | 626.21 | 3671.64 | 4.89 | 34996 |
| HE1N | 3 | 8 | 32 | 16850.53 | 402.77 | 307.76 | 1659.24 | 3.25 | 33540 |
| HE1N | 3 | 8 | 64 | 2807.49 | 197.57 | 75.14 | 352.67 | 2.31 | 33591 |
| HE1N | 3 | 8 | 128 | 3701.99 | 585.24 | 340.2 | 1834.74 | 4.59 | 37663 |
| HE1N | 3 | 16 | 64 | 18828.28 | 415.87 | 309.18 | 1696.96 | 3.36 | 35064 |
| HE1N | 3 | 16 | 128 | 4672.69 | 474.73 | 315.1 | 1708.43 | 3.41 | 38737 |
| HE2 | 2 | 32 | - | 122.25 | 525.64 | 103.38 | 75.11 | 1.86 | 784 |
| HE2 | 2 | 64 | - | 116.27 | 549.24 | 118.68 | 84.28 | 1.91 | 1308 |
| HE2 | 2 | 128 | - | 131.78 | 624.37 | 120.37 | 168.72 | 2.12 | 1482 |
| HE2 | 3 | 32 | - | 152.55 | 560.01 | 103.56 | 111.55 | 2.14 | 960 |
| HE2 | 3 | 64 | - | 180.11 | 650.07 | 122.14 | 211.68 | 2.24 | 1650 |
| HE2 | 3 | 128 | - | 214.22 | 1462.87 | 452.92 | 2423.63 | 2.81 | 2399 |
| HE2 | 4 | 32 | - | 160.11 | 658.37 | 130.34 | 219.3 | 2.43 | 1687 |
| HE2 | 4 | 64 | - | 257.11 | 1446.8 | 448.06 | 2432.22 | 2.87 | 2610 |
| HE2 | 4 | 128 | - | 1102.95 | 1921.06 | 601.83 | 3147.91 | 3.84 | 4200 |
| HE2N | 2 | 1 | 32 | 154.46 | 677.64 | 121.81 | 197.3 | 2.32 | 1149 |
| HE2N | 2 | 1 | 64 | 360.2 | 1464.05 | 422.87 | 2256.1 | 2.54 | 2889 |
| HE2N | 2 | 1 | 128 | 869.88 | 1992.68 | 511.34 | 2896.89 | 3.43 | 3257 |
| HE2N | 2 | 8 | 32 | 384.53 | 1472.49 | 433.05 | 2275.24 | 2.71 | 2515 |
| HE2N | 2 | 8 | 64 | 196.91 | 713.34 | 140.85 | 290.54 | 2.33 | 1154 |
| HE2N | 2 | 8 | 128 | 363.02 | 1803.51 | 503.41 | 2789.1 | 3.54 | 2740 |
| HE2N | 2 | 16 | 64 | 411.43 | 764.95 | 146.27 | 336.36 | 2.5 | 1833 |
| HE2N | 2 | 16 | 128 | 228.27 | 1660.48 | 466.07 | 2528.7 | 2.87 | 2565 |
| HE2N | 3 | 1 | 32 | 331.43 | 1553.09 | 446.9 | 2450.82 | 2.86 | 5360 |
| HE2N | 3 | 1 | 64 | 1197.62 | 5841.29 | 3120.64 | 16400 | 4.99 | 8129 |
| HE2N | 3 | 1 | 128 | 2805.34 | 12600 | 7193.61 | 40300 | 8.25 | 12565 |
| HE2N | 3 | 8 | 32 | 2658.76 | 5828.94 | 3317.31 | 17900 | 5.36 | 8676 |
| HE2N | 3 | 8 | 64 | 1047.95 | 1852.06 | 537.16 | 2996.51 | 3.59 | 3753 |
| HE2N | 3 | 8 | 128 | 7375.87 | 7292.07 | 4344.95 | 19500 | 7.31 | 11014 |
| HE2N | 3 | 16 | 64 | 850.94 | 6375.15 | 3359.67 | 18300 | 5.27 | 10942 |
| HE2N | 3 | 16 | 128 | 7057.17 | 6986.84 | 3358.11 | 17900 | 5.45 | 6026 |

which is a single bit. For example, it should be noted, that even the practical CryptDB [65], which is only additively homomorphic, enciphers a 32-bit integer as a 2048-bit ciphertext. Our schemes will produce ciphertext of similar size, if high security is required. However, if the security is only intended to prevent casual snooping, rather than a determined cryptographic attack, the ciphertext size can be reduced. Observe that the parameters of the system will change for each computation, so a sustained attack has constantly to re-learn these parameters. Of course, if the attacker is able to export data for off-line cryptanalysis, only high security suffices.

The schemes presented in sections 3 and 4 extend to a hierarchy of systems, HE$k$, with increasing levels of security. These are presented in section 5 and may be investigated further in future work. As stated in section 6, we can extend HE2NCRT to $k$-vectors to create a HE$k$NCRT scheme. The discussion will appear in future work. We also showed that our HE$k$ system can be extended to an FHE system. We may also investigate several enhancements of this FHE system. First, we

could implement a packed ciphertext optimisation for our scheme where an operation is performed on a vector of ciphertexts rather than a single ciphertext [75]. We can also investigate an improvement to address circuit privacy [19]. Finally, we can investigate applying the Chinese Remainder Theorem secret sharing method employed in HE2NCRT (section 6) to our fully homomorphic scheme (section 7).

We have implemented and evaluated the schemes HE1, HE1N, HE2 and HE2N as part of an SSCC system as discussed in section 8. Our results are extremely favourable by comparison with existing methods. In some cases, they outperform those methods by a factor of 1000. We have also performed extensive experiments with large data sets (approximately 100 million inputs) and shown that the increase in time to perform encryption and arithmetic operations grows sublinearly in the number of inputs. This clearly demonstrates the practical applicability of our schemes. Furthermore, our MapReduce job execution times remain low even when using the largest set of parameters for HE2N. We believe that this demonstrates the advantages of these schemes for encrypted computations on fixed-point data in the cloud.

To conclude, we believe that the work in this paper, has achieved our goal of providing a homomorphic encryption scheme over the integers suitable for SSCC. In particular, our work significantly outperforms related work [65,76,77] with regard to the running time of arithmetic operations on encrypted data.

**Acknowledgements** This work was supported in part by Engineering and Physical Sciences Research Council (EPSRC) research grants EP/I028099/1 and EP/M004953/1 and National Key Research and Development Program of China research grant 2016YFB1000103. Experimentation conducted on Microsoft Azure was supported by a Microsoft Azure for Research sponsorship.

**Compliance with Ethical Standards**
Funding: This study was funded by Engineering and Physical Sciences Research Council (EP/I028099/1 & EP/M004953/1) and National Key Research and Development Program of China (2016YFB1000103). Usage of Microsoft's Azure cloud was funded by a Microsoft Azure for Research sponsorship.
Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

## References

1. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A Survey on Homomorphic Encryption Schemes: Theory and Implementation. ACM Computing Surveys **51**(4), 79:1–79:35 (2018). DOI 10.1145/3214303
2. Aggarwal, N., Gupta, C., Sharma, I.: Fully Homomorphic Symmetric Scheme Without Bootstrapping. In: Proceedings of 2014 International Conference on Cloud Computing and Internet of Things (CCIOT 2014), pp. 14–17 (2014). DOI 10.1109/CCIOT.2014.7062497
3. Aguilar-Melchor, C., Killijian, M.O., Lefebvre, C., Lepoint, T., Ricosset, T.: A Comparison of Open-Source Homomorphic Libraries With Multi-Precision Plaintext Moduli (2016). URL `https://wheat2016.lip6.fr/ricosset.pdf`. WHEAT 2016
4. Amazon Web Services, Inc.: Amazon EMR (2018). URL `http://aws.amazon.com/elasticmapreduce/`
5. Aumasson, J.P.: On the pseudo-random generator ISAAC. Cryptology ePrint Archive, Report 2006/438 (2006). `https://eprint.iacr.org/2006/438`
6. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A Concrete Security Treatment of Symmetric Encryption. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS 1997), pp. 394–403. IEEE (1997). DOI 10.1109/SFCS.1997.646128
7. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations Among Notions of Security for Public-Key Encryption Schemes. In: Proceedings of the 18th Annual Cryptology Conference (CRYPTO 1998), pp. 26–45. Springer-Verlag (1998). DOI 10.1007/BFb0055718
8. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of Garbled Circuits. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS 2012), pp. 784–796. ACM (2012). DOI 10.1145/2382196.2382279
9. Bellare, M., Rogaway, P.: Introduction to Modern Cryptography (2005). Lecture Notes
10. Berlekamp, E.R.: Factoring Polynomials Over Large Finite Fields. Mathematics of Computation **24**(111), 713–735 (1970)
11. Bernstein, D.J.: Post-Quantum Cryptography, chap. Introduction to post-quantum cryptography, pp. 1–14. Springer-Verlag (2009). DOI 10.1007/978-3-540-88702-7_1
12. Bogos, S., Gaspoz, J., Vaudenay, S.: Cryptanalysis of a Homomorphic Encryption Scheme. Cryptology ePrint Archive, Report 2016/775 (2016). `https://eprint.iacr.org/2016/775`
13. Bogos, S., Gaspoz, J., Vaudenay, S.: Cryptanalysis of a homomorphic encryption scheme. Cryptography and Communications **10**(1), 27–39 (2018). DOI 10.1007/s12095-017-0243-8
14. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography (2015). Draft 0.2
15. Bonte, C., Bootland, C., Bos, J.W., Castryck, W., Iliashenko, I., Vercauteren, F.: Faster Homomorphic Function Evaluation using Non-Integral Base Encoding. Cryptology ePrint Archive, Report 2017/333 (2017). `https://eprint.iacr.org/2017/333`
16. Bos, J.W., Castryck, W., Iliashenko, I., Vercauteren, F.: Privacy-friendly Forecasting for the Smart Grid using Homomorphic Encryption and the Group Method of Data Handling. Cryptology ePrint Archive, Report 2016/1117 (2016). `https://eprint.iacr.org/2016/1117`
17. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: R. Safavi-Naini, R. Canetti (eds.) Proceedings of the 32nd Annual Cryptology Conference (CRYPTO 2012), pp. 868–886. Springer-Verlag (2012). DOI 10.1007/978-3-642-32009-5_50
18. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In: Proceedings of the 3rd Conference on Innovations in

Theoretical Computer Science (ITCS 2012), pp. 309–325. ACM (2012). DOI 10.1145/2090236.2090262
19. Brakerski, Z., Vaikuntanathan, V.: Efficient Fully Homomorphic Encryption from (Standard) LWE. In: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011), pp. 97–106. IEEE (2011). DOI 10.1109/FOCS.2011.12
20. Brakerski, Z., Vaikuntanathan, V.: Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: P. Rogaway (ed.) Proceedings of the 31st Annual Cryptology Conference (CRYPTO 2011), pp. 505–524. Springer-Verlag (2011). DOI 10.1007/978-3-642-22792-9_29
21. Catalano, D., Fiore, D.: Boosting Linearly-Homomorphic Encryption to Evaluate Degree-2 Functions on Encrypted Data. Cryptology ePrint Archive, Report 2014/813 (2014). https://eprint.iacr.org/2014/813
22. Catalano, D., Fiore, D.: Using Linearly-Homomorphic Encryption to Evaluate Degree-2 Functions on Encrypted Data. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015), pp. 1518–1529. ACM (2015). DOI 10.1145/2810103.2813624
23. Chen, L., Ben, H., Huang, J.: An Encryption Depth Optimization Scheme for Fully Homomorphic Encryption. In: Proceedings of the 2014 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI 2014), pp. 137–141. IEEE (2014). DOI 10.1109/IIKI.2014.35
24. Chen, Y., Nguyen, P.Q.: Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers. In: D. Pointcheval, T. Johansson (eds.) Proceedings of the 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2012), pp. 502–519. Springer-Verlag (2012). DOI 10.1007/978-3-642-29011-4_30
25. Cheon, J.H., Coron, J., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch Fully Homomorphic Encryption over the Integers. In: T. Johansson, P.Q. Nguyen (eds.) Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2013), pp. 315–335. Springer-Verlag (2013). DOI 10.1007/978-3-642-38348-9_20
26. Cheon, J.H., Kim, J., Lee, M.S., Yun, A.: CRT-Based Fully Homomorphic Encryption Over the Integers. Information Sciences **310**, 149–162 (2015). DOI 10.1016/j.ins.2015.03.019
27. Cheon, J.H., Stehlé, D.: Fully Homomophic Encryption over the Integers Revisited. In: E. Oswald, M. Fischlin (eds.) Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2015), Part I, pp. 513–536. Springer-Verlag (2015). DOI 10.1007/978-3-662-46800-5_20
28. Cohn, H., Heninger, N.: Approximate common divisors via lattices. In: Proceedings of the 10th Algorithmic Number Theory Symposium (ANTS-X), vol. 1, pp. 271–293. Mathematical Sciences Publishers (2012). DOI 10.2140/obs.2013.1.271
29. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology **10**(4), 233–260 (1997). DOI 10.1007/s001459900030
30. Coron, J., Lepoint, T., Tibouchi, M.: Scale-Invariant Fully Homomorphic Encryption over the Integers. In: H. Krawczyk (ed.) Proceedings of the 17th International Conference on Practice and Theory in Public-Key Cryptography (PKC 2014), pp. 311–328. Springer-Verlag (2014). DOI 10.1007/978-3-642-54631-0_18
31. Coron, J., Mandal, A., Naccache, D., Tibouchi, M.: Fully Homomorphic Encryption over the Integers with Shorter Public Keys. In: P. Rogaway (ed.) Proceedings of the 31st Annual Cryptology Conference (CRYPTO 2011), pp. 487–504. Springer-Verlag (2011). DOI 10.1007/978-3-642-22792-9_28
32. Coron, J., Naccache, D., Tibouchi, M.: Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. In: D. Pointcheval, T. Johansson (eds.) Proceedings of the 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2012), pp. 446–464. Springer-Verlag (2012). DOI 10.1007/978-3-642-29011-4_27
33. CryptoExperts: FV-NFLib. URL https://github.com/CryptoExperts/FV-NFLlib
34. Dautelle, J.M.: JScience (2014). URL http://jscience.org
35. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers. In: Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT 2010), pp. 24–43. Springer-Verlag (2010). DOI 10.1007/978-3-642-13190-5_2
36. van Dijk, M., Juels, A.: On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing. In: Proceedings of the 5th USENIX Workshop on Hot Topics in Security (HotSec 2010), pp. 1–8. USENIX Association (2010). URL https://www.usenix.org/legacy/events/hotsec10/tech/full_papers/vanDijk.pdf
37. Ducas, L., Micciancio, D.: FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In: E. Oswald, M. Fischlin (eds.) Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2015), Part I, pp. 617–640. Springer-Verlag (2015). DOI 10.1007/978-3-662-46800-5_24
38. Ducas, L., Micciancio, D.: FHEW (2017). URL https://github.com/lducas/FHEW
39. Dyer, J., Dyer, M., Xu, J.: Practical Homomorphic Encryption Over the Integers for Secure Computation in the Cloud. In: M. O'Neill (ed.) Proceedings of the 16th IMA International Conference on Cryptography and Coding (IMACC 2017), *Lecture Notes in Computer Science*, vol. 10655, pp. 44–76. Springer-Verlag (2017). DOI 10.1007/978-3-319-71045-7_3
40. ElGamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). DOI 10.1109/TIT.1985.1057074
41. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-Preserving Face Recognition. In: I. Goldberg, M.J. Atallah (eds.) Proceedings of the 9th International Symposium on Privacy Enhancing Technologies (PETS 2009), pp. 235–253. Springer-Verlag (2009). DOI 10.1007/978-3-642-03168-7_14
42. Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144 (2012). https://eprint.iacr.org/2012/144
43. Galbraith, S.D., Gebregiyorgis, S.W., Murphy, S.: Algorithms for the approximate common divisor problem.

LMS Journal of Computation and Mathematics **19**(A), 58–72 (2016). DOI 10.1112/S1461157016000218
44. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009), pp. 169–178. ACM (2009). DOI 10.1145/1536414.1536440
45. Goldreich, O., Micali, S., Wigderson, A.: How to Play ANY Mental Game. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987), pp. 218–229. ACM (1987). DOI 10.1145/28395.28420
46. Goldwasser, S., Kalai, Y., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable Garbled Circuits and Succinct Functional Encryption. In: Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013), pp. 555–564. ACM (2013). DOI 10.1145/2488608.2488678
47. Goldwasser, S., Micali, S.: Probabilistic Encryption. Journal of Computer and System Sciences **28**(2), 270 – 299 (1984). DOI 10.1016/0022-0000(84)90070-9
48. Halevi, S., Shoup, V.: HELib. URL `https://github.com/shaih/HElib`
49. Halevi, S., Shoup, V.: Bootstrapping for HELib. In: E. Oswald, M. Fischlin (eds.) Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2015), pp. 641–670. Springer-Verlag (2015). DOI 10.1007/978-3-662-46800-5_25
50. Howgrave-Graham, N.: Approximate Integer Common Divisors. In: Revised Papers from the International Conference on Cryptography and Lattices (CaLC 2001), pp. 51–66. Springer-Verlag (2001). DOI 10.1007/3-540-44670-2_6
51. Hrubeš, P., Yehudayoff, A.: Arithmetic Complexity in Algebraic Extensions. Theory of Computing **7**, 119–129 (2011)
52. Jenkins, B.: ISAAC: a fast cryptographic random number generator (1996). URL `http://burtleburtle.net/bob/rand/isaacafa.html`
53. Joye, M., Libert, B.: Efficient Cryptosystems from $2^k$-th Power Residue Symbols. In: Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2013), pp. 76–92. Springer-Verlag (2013). DOI 10.1007/978-3-642-38348-9_5
54. Kipnis, A., Hibshoosh, E.: Efficient Methods for Practical Fully Homomorphic Symmetric-key Encrypton, Randomization and Verification. Cryptology ePrint Archive, Report 2012/637 (2012). `https://eprint.iacr.org/2012/637`
55. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., Te Riele, H., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA Modulus. In: Proceedins of the 30th Annual Cryptology Conference (CRYPTO 2010), pp. 333–350. Springer-Verlag (2010). DOI 10.1007/978-3-642-14623-7_18
56. Laine, K., Chen, H., Player, R., Wernsing, J., Naehrig, M., Dowlin, N., Cetin, G., Xia, S., Rindal, P.: Simple Encrypted Arithmetic Library - SEAL (2017). URL `https://sealcrypto.codeplex.com/`
57. Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can Homomorphic Encryption Be Practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop (CCSW 2011), pp. 113–124. ACM (2011). DOI 10.1145/2046660.2046682
58. Massey, J.L.: Guessing and Entropy. In: Proceedings of 1994 IEEE International Symposium on Information Theory (ISIT 1994), p. 204. IEEE (1994). DOI 10.1109/ISIT.1994.394764
59. Microsoft: HDInsight (2018). URL `http://azure.microsoft.com/en-gb/services/hdinsight/`
60. Microsoft: Microsoft Azure for Research (2018). URL `https://www.microsoft.com/en-us/research/academic-program/microsoft-azure-for-research/`
61. Moshkovitz, D.: An Alternative Proof of the Schwartz-Zippel Lemma. In: Electronic Colloquium on Computational Complexity (ECCC), p. 96 (2010)
62. Nuida, K., Kurosawa, K.: (Batch) Fully Homomorphic Encryption over Integers for Non-Binary Message Spaces. Cryptology ePrint Archive, Report 2014/777 (2014). `https://eprint.iacr.org/2014/777`
63. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: J. Stern (ed.) Proceedings of the 18th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 1999), pp. 223–238. Springer-Verlag (1999). DOI 10.1007/3-540-48910-X_16
64. Pisa, P.S., Abdalla, M., Duarte, O.C.M.B.: Somewhat homomorphic encryption scheme for arithmetic operations on large integers. In: Proceedings of the 2012 Global Information Infrastructure and Networking Symposium (GIIS 2012), pp. 1–8. IEEE (2012). DOI 10.1109/GIIS.2012.6466769
65. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: Protecting Confidentiality with Encrypted Query Processing. In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011), pp. 85–100. ACM (2011). DOI 10.1145/2043556.2043566
66. Rabin, M.O.: Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Tech. Rep. MIT/LCS/TR-212, MIT (1979)
67. Ramaiah, Y.G., Kumari, G.V.: Efficient Public Key Homomorphic Encryption over Integer Plaintexts. In: Proceedings of the 2012 International Conference on Information Security and Intelligent Control, pp. 123–128 (2012). DOI 10.1109/ISIC.2012.6449723
68. Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005), pp. 84–93. ACM (2005). DOI 10.1145/1060590.1060603
69. Ricosset, T.: HElib-MP. URL `https://github.com/tricosset/HElib-MP`
70. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On Data Banks and Privacy Homomorphisms. Foundations of Secure Computation **4**(11), 169–180 (1978)
71. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM **21**(2), 120–126 (1978). DOI 10.1145/359340.359342
72. Schafer, R.D.: An Introduction to Nonassociative Algebras, vol. 22. Dover (1966)
73. Scharle, T.W.: Axiomatization of Propositional Calculus With Sheffer Functors. Notre Dame Journal of Formal Logic **6**(3), 209–217 (1965). DOI 10.1305/ndjfl/1093958259
74. Smart, N.P., Vercauteren, F.: Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In: Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography (PKC 2010), pp. 420–443. Springer-Verlag (2010). DOI 10.1007/978-3-642-13013-7_25

75. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Designs, Codes and Cryptography **71**(1), 57–81 (2014). DOI 10.1007/s10623-012-9720-4
76. Stephen, J.J., Savvides, S., Seidel, R., Eugster, P.: Practical Confidentiality Preserving Big Data Analysis. In: Proceedings of the 6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14). USENIX Association (2014). URL `https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/stephen`
77. Tetali, S.D., Lesani, M., Majumdar, R., Millstein, T.: MrCrypt: Static Analysis for Secure Cloud Computations. In: Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA 2013), pp. 271–286. ACM (2013). DOI 10.1145/2509136.2509554
78. Thomson, I.: Microsoft researchers smash homomorphic encryption speed barrier. The Register (2016). URL `https://www.theregister.co.uk/2016/02/09/researchers_break_homomorphic_encryption/`
79. Varia, M., Yakoubov, S., Yang, Y.: HETest: A Homomorphic Encryption Testing Framework. Cryptology ePrint Archive, Report 2015/416 (2015). `https://eprint.iacr.org/2015/416`
80. Vivek, S.: Homomorphic Encryption API Software Library. URL `http://heat-h2020-project.blogspot.co.uk/2017/02/homomorphic-encryption-api-software.html`
81. Vizár, D., Vaudenay, S.: Cryptanalysis of Chosen Symmetric Homomorphic Schemes. Studia Scientiarum Mathematicarum Hungarica **52**(2), 288–306 (2015). DOI 10.1556/012.2015.52.2.1311
82. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer (1999)
83. Wang, D., Guo, B., Shen, Y., Cheng, S.J., Lin, Y.H.: A faster fully homomorphic encryption scheme in big data. In: Proceedings of the 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA 2017), pp. 345–349 (2017). DOI 10.1109/ICBDA.2017.8078836
84. Yu, A., Lai, W.L., Payor, J.: Efficient Integer Vector Homomorphic Encryption (2015). URL `https://courses.csail.mit.edu/6.857/2015/files/yu-lai-payor.pdf`
85. Zhou, H., Wornell, G.: Efficient Homomorphic Encryption on Integer Vectors and its Applications. In: Proceedings of the 2014 Information Theory and Applications Workshop (ITA 2014), pp. 1–9. IEEE (2014). DOI 10.1109/ITA.2014.6804228