Practical Homomorphic Encryption Over the Integers

James Dyer[*1], Martin Dyer[†2], and Jie Xu[‡2]

[1]School of Computer Science, University of Manchester
[2]School of Computing, University of Leeds

27th February 2017

Abstract

We present novel homomorphic encryption schemes for integer arithmetic, intended for use in secure single-party computation in the cloud. These schemes are capable of securely computing only low degree polynomials homomorphically, but this appears sufficient for most practical applications. In this setting, our schemes lead to practical key and ciphertext sizes. We present a sequence of generalisations of our basic schemes, with increasing levels of security, but decreasing practicality. We have evaluated the first four of these algorithms by computing a low-degree inner product. The timings of these computations are extremely favourable. Finally, we use our ideas to derive a fully homomorphic system, which appears impractical, but can homomorphically evaluate arbitrary Boolean circuits.

1 Introduction

With services like Amazon's Elastic MapReduce and Microsoft's HDInsight 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 [21] 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) seems to offer a solution to the SSCC problem. First defined by Rivest et al. [43] 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 Dec is the decryption function. One can easily see that HE would satisfy some of the requirements for secure computation in the cloud. A *somewhat HE* scheme (SWHE) is an HE scheme which is HE for only limited imputs and functions.

Fully HE (FHE) is an HE scheme that is homomorphic for all f. This was first realised by Gentry in 2009 [24], and appears to be the ideal HE scheme. However, despite the clear advantages of FHE, and many

---

[*]james.dyer@postgrad.manchester.ac.uk

[†]m.e.dyer@leeds.ac.uk

[‡]j.xu@leeds.ac.uk

significant advances [10–13], it remains largely impractical. The two implementations of recent FHE schemes, HELib [29] and FHEW [22], both perform very poorly in comparison with operations on unencrypted data, in their running time and space requirements. It is reported that a HELib implementation of the AES-128 circuit processed inputs in just over four minutes [30]. Similarly, FHEW processed a single homomorphic NAND operation followed by a re-encryption in 0.69s and using 2.2GB of RAM. The paper [36] attempted to assess the practicality of one of the underlying SWHE schemes[12], but with no positive conclusion.

Therefore, we take the view in this paper that only SWHE is currently of practical interest. Our goal is to develop new SWHE schemes which are practically useful, and which we have implemented, though we conclude the paper by showing that our ideas can be used to develop a (fairly impractical) FHE scheme.

1.1 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. For the most part, 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, and the non-numeric fields would be 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 could be casual, simply displaying an uninvited interest, or malicious, intending to use the data for the attacker's own purposes.

To obtain the required data privacy, the client's function will be computed homomorphically, on an encrypted version of the data. The client encrypts the source data using a secret key and uploads the encrypted data to the cloud, along 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 then returned to the client, who decrypts it using the secret key, and obtains the output of the original computation.

In this scenario, we observe that the source data is never exposed in the cloud, but encryptions of the source data 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 on the data, 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 on this data. However, snoopers have no access to the secret key, so cannot make encryptions of their own to deduce the secret key.

1.2 Definitions and Notation

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

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

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 $\mathcal{K}$ and whose range is the ciphertext space $\mathcal{C}$.

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 $\mathcal{K}$ and whose range is the plaintext space $\mathcal{M}$.

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

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

$m, m_1, m_2, \ldots$ denote plaintext values. Similarly, $c, c_1, c_2, \ldots$ denote ciphertext values.

If $k^* = \binom{k+1}{2}$, $\boldsymbol{v}_\star = [v_1 \ v_2 \ \ldots \ v_{k^*}]^T$ denotes a k^*-vector which augments the k-vector $\boldsymbol{v} = [v_1 \ v_2 \ \ldots \ v_k]^T$ 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.)

e_i denotes the ith unit vector ($i = 1, 2, \ldots$), with size determined by the context.

$[x, y]$ denotes the integers between x and y inclusive.

$[x, y)$ denotes $[x, y] \setminus \{y\}$.

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

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

Polynomial time or space means time or space polynomial in the security parameter λ.

An *arithmetic circuit* Φ over the ring R in variables $X = \{x_1, \ldots, x_n\}$ is a directed acyclic graph with every vertex (*gate*) having in-degree either two or zero. Every vertex of in-degree 0 is labelled either by a variable in X or by an element of R. Each other vertex in Φ has in-degree two and is labelled by either $\times$ or $+$. Every vertex of out-degree 0 in Φ computes a polynomial in $\mathsf{R}[X]$ in the obvious manner. We refer to the directed edges in the acyclic graph as *wires*. The *depth* of Φ is the length of the longest directed path in it. (See [32].)

A *Boolean circuit* Φ is defined similarly to an arithmetic circuit. Every vertex of in-degree 0 is labelled either by a variable in X or an element of $\{0, 1\}$. Each other vertex in Φ has in-degree two and is labelled by a binary Boolean function. Then every vertex of out-degree 0 in Φ computes some Boolean function of the inputs. Note that any finite computation can be represented as a Boolean circuit. (See [49].)

1.3 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 entropy $\quad H_1(X) = -\sum_{i=0}^{M-1} \xi_i \lg \xi_i$,

Collision entropy $\quad H_2(X) = -\lg\left(\sum_{i=0}^{M-1} \xi_i^2\right)$,

Min entropy $\quad\quad H_\infty(X) = -\lg\left(\max_{i=0}^{M-1} \xi_i\right)$.

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 ρ, 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 polynomial P of degree d on these inputs. A secure client A selects an instance $\mathcal{E}_K$ of the encryption algorithm $\mathcal{E}$ using the secret parameter set K. A encrypts the n inputs by computing $c_i = \mathcal{E}_K(m_i)$, 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 computing 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

$$P(m_1, m_2, \ldots, m_n) = \mathcal{E}_K^{-1}(P'(c_1, c_2, \ldots, c_n)).$$

A snooper is only able to inspect $c_1, c_2, \ldots, c_n$, the function P', the computation of $P'(c_1, c_2, \ldots, c_n)$, including subcomputations and working data, and $P'(c_1, c_2, \ldots, c_n)$ itself.

Our encryption schemes are essentially symmetric key encryption, though there is no key distribution problem. The public parameters of our schemes are exposed to the cloud, but they do not provide an encryption oracle.

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 μ of guesses before making a correct guess satisfies $\mu \geq 2^\rho$. Massey [37] gave a corresponding result in terms of the Shannon entropy $H_1(X)$.

It follows similarly that the probability of any correct guess in $2^{\rho/2}$ guesses is at most $2^{-\rho/2}$. This bound holds if we need only to guess any one of n inputs, $m_1, m_2, \ldots, m_n$, even if these inputs are not independent. Therefore, if ρ is large enough, a brute force attack is infeasible.

3

Recall that, in our model, known plaintext attack (KPA) is possible only by brute force, and not through being given a sample of plaintext, ciphertext pairs.

We do not regard chosen plaintext attack (CPA) or chosen ciphertext attack (CCA) as being relevant to our model. Since $\mathcal{E}_K$ is never exposed in the cloud, there is no realistic analogue of an encryption or decryption oracle, as required by these attacks. Of course, in public key encryption, an encryption algorithm is publicly available as part of the system, so CPA must be forestalled. We note that, following [4], it is common in studying symmetric key encryption to suppose that defence to CPA or CCA is necessary. While this may provide a stronger notion of security, it seems hard to justify. Both [3] and [8] provide examples which are intended to justify this convention. However, these examples are unconvincing, and seem to have little practical importance. Nevertheless, since it is not difficult to do so, we show that the "N" variants of our HE schemes below resist 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. This assumption seems universal in the existing literature on HE. However, "garbled circuits" [5, 25] are a possible solution to this problem, if the threat is significant.

Finally, we note that our model of SSCC is very similar to the model of *private single-client computing*, presented in [21] along with an example application.

1.4   Our Results

We describe novel practical HE schemes for the encryption of integers, to be employed in a SSCC system inspired by CryptDB [40]. CryptDB is an HE scheme where encryption depends on the operation to be performed. CryptDB encrypts integers using the Paillier cryptosystem [39] which allows for homomorphic addition. Similar systems ([46, 47]) use Paillier and ElGamal [23] to support addition and multiplication, respectively. The "unpadded" versions of these schemes are used, which may not be secure under CPA [26], reducing any possible advantage of a public-key system. However, these schemes do not support both addition and multiplication. To perform an inner product, say, requires re-encrypting the data once the multiplications have been performed so that the additions can then be performed. In a SSCC system, this would require shipping the data back to the initiator for re-encryption, creating a significant overhead. To avoid this problem, we aim for an HE scheme for integers supporting both addition and multiplication.

Our HE scheme over the integers is inspired by the SWHE scheme of van Dijk et al. [20] (which we denote DGHVS) that is used as the basis for their public-key system (denoted as DGHV in [18]). As in their system, we add multiples of integers to the plaintext to produce a ciphertext. However, DGHVS supports only arithmetic mod 2, and we generalise this to larger moduli.

In the section above, we showed that the input data must have sufficient entropy to negate brute force attacks. If the data lacks sufficient entropy, we will introduce more entropy in two ways. The first is to add random "noise" of sufficient entropy to the ciphertext to "mask" the plaintext. This approach is employed in DGHV and in sections 2.2 and 3.2. In our schemes we add a random multiple (from 0 to κ) of a large integer, κ, to the ciphertext, such that $m_i < \kappa$, for all $i \in [1, N]$. If the entropy of the original data was ρ, once transformed it is $\rho + \lg \kappa$. Therefore, if κ is large enough we can ensure that our data has sufficient entropy. However, there is a downside. To prevent the noise term growing so large that the cipherext can no longer be decrypted successfully, we are restricted to computing polynomials of low enough degree.

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 as in DGHVS. Addition and multiplication of ciphertexts are simple transformations of the ciphertexts using vector and matrix algebra. The basic case $k = 1$ is described in section 2.1. Then we can increase the entropy k-fold by creating a k-vector ciphertext. This is because we need to guess k plaintexts to successfully break the system. Assuming that the inputs $m_1, m_2, \ldots, m_n$ are chosen independently from $\mathcal{D}$, and the entropy of inputs is ρ, then 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, there will be sufficient entropy to prevent brute force attack. Note that the assumption of independence among $m_1, m_2, \ldots, m_n$ can easily be relaxed, to allow some correlation, but we will not discuss the details here. The upside is that some cryptanalytic methods applicable in the case $k = 1$ 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, in comparison with the case $k = 1$. For very large k, this probably renders the

methods impractical. Therefore, we consider the case $k = 2$ in some detail in section 3.1, before describing the general case in section 4.1.

Our work here only aims to support integer arithmetic. Other operations, like sorting, require different HE schemes, which we will consider elsewhere. In the integer arithmetic case, a system for computing low-degree polynomials seems to suffice for most practical applications. (See [36]). To this end, we will consider practically implementable values for the parameters of the cryptosystems.

1.5   Related Work

FHE schemes start by devising a SWHE scheme which supports only homomorphic addition and multiplication, so the computation is to evaluate an arithmetic circuit on encryptions to which random "noise" has been edded. A general computation is represented as an arithmetised Boolean circuit [2]. As ciphertexts are added and multiplied during the computation, the "noise" grows until there comes a point where the plaintext cannot be uniquely recovered from the ciphertext. Therefore the arithmetic circuit must be of limited depth, to prevent this "noise" growing too large. If the circuit is sufficiently shallow, this SWHE scheme can be used in its own right, e.g. Boneh et al. [9].

The SWHE system is transformed into an FHE scheme by a process of re-encryption where, once the noise grows too large, the ciphertext is re-encrypted, thereby allowing computation to proceed. This re-encryption is performed homomorphically, using a circuit shallow enough that the noise does not grow too large. This circuit necessarily contains information about the private key, which must be suitably hidden. This re-encryption reduces the noise in the ciphertext, and allows circuits of arbitrary depth to be computed homomorphically.

Our scheme is inspired by that of van Dijk et al. [20]. In their paper they produce an FHE scheme over the integers, where a simple SWHE scheme is "bootstrapped" to FHE. van Dijk et al. take a simple symmetric scheme where an plaintext bit m is encrypted as $c = m + 2r + pq$, where the secret key p is an odd η-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 may be regarded as a generalisation of this.

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 calculated as $c = m + 2r + 2 \sum_{i \in S} x_i \mod x_0$. The ciphertext is decrypted as previously described. We could extend our HEkN schemes here to a public key variant, using a similar device. However, we will not do so, since public key systems appear to have little application to our model.

van Dijk et al. "bootstrap" their public key system to an FHE scheme, using Gentry's approach [24]. In this case, the bootstrapping is done by homomorphically simulating a division by p, thus obtaining an encryption of $c \bmod p$ which can be used to continue the computation. Our FHE proposal below is based on different principles.

In [18], Coron et al. 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.

The above FHE schemes represent a major theoretical achievement. However, they appear impractical for computations on large data sets, in terms of both running time and storage requirements.

Therefore, the direction of our work is similar to [36]. The authors implement the SWHE scheme from [12]. However, they give results only for degree two polynomials. Our schemes seem capable of computing somewhat higher degree polynomials for practical key and ciphertext sizes.

Recent work on *functional encryption* [27, 28] 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 [27, 28] do not seem likely to be of practical interest in the foreseeable future.

The symmetric MORE scheme [34] and its derivative [50] uses linear transformations, as does our scheme HEk in a different way. These systems have been shown [48] to be insecure against KPA, at least as originally proposed. However, whether KPA is practically relevant in context is moot.

We also note the work of Cheon et al. [15]. They use the Chinese Remainder Theorem (CRT) in an FHE system. We make use of the CRT in our scheme HE2NCRT below (section 5). However, our construction

differs significantly from theirs.

We should note that the encryption of the Boolean circuits in our fully homomorphic system (section 6) has similarities to Yao's "garbled circuits" [5, 25].

1.6 Roadmap

We present our initial homomorphic scheme in section 2 in two variants, HE1 and HE1N. HE1 (section 2.1) is suitable for integers distributed with sufficient entropy. HE1N (section 2.2) deals with integers not distributed with sufficient entropy, by adding an additional "noise" term.

Section 3 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. Again, HE2 (section 3.1) deals with integers of sufficient entropy, HE2N (section 3.2) with integers without the required entropy. We describe this in some detail, since it appears to be practically useful, and is the simplest version of our general scheme.

In section 4, we generalise HE2 and HE2N from 2-vectors to k-vectors, for arbitrary k, in the scheme HEk, with noisy variant HEkN. These schemes may also be practical for small enough k.

In section 5, we present an extension of HE2N, HE2NCRT, which uses the CRT to distribute the computation.

In section 6, we discuss how HEk can be transformed into an FHE scheme for large enough k, though the resulting scheme seems only to be of theoretical interest.

In section 7 we describe extensive experimentation with the schemes, and finally, in section 8, we give our conclusions.

2 Initial Homomorphic Scheme

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

2.1 Sufficient Entropy (HE1)

We have integer inputs $m_1, m_2, \ldots, m_n \in [0, M)$. (Negative integers can be handled as in van Dijk et al. [20], 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 ρ, where ρ is large enough, as discussed in section 1.3 above. Our HE scheme is the system (KeyGen, Enc, Dec, Add, Mult).

Let λ be a large enough security parameter, measured in bits. Let p and q be suitably 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 λ must be large enough to negate direct factorisation of pq (see [35]), and the relative values of p and q are chosen to negate Coppersmith's attack [17]. 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. (In many applications, a smaller value of p may suffice). Our function KeyGen 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.

We can easily set the parameters to practical values. If $n \approx \sqrt{M}$, $M \approx 2^\rho$ then we may take $\lambda \approx 3d\rho/2$ and $\eta \approx 3d\lambda/2 - \lambda$ (see appendix B). For, example, if $\rho = 32$, $d = 4$, we can take any $\lambda > 192$, $\eta > 960$.

We encrypt a plaintext integer m as

$$\mathsf{Enc}(m, p) = m + rp \mod pq$$

where $r \xleftarrow{\$} [1, q)$.

We decrypt the ciphertext c by

$$\mathsf{Dec}(c, p) = c \mod p$$

The sum modulo pq of two ciphertexts, $c = m + rp$ and $c' = m' + r'p$, is

$$\mathsf{Add}(c, c') = c + c' \mod pq = m + m' + (r + r')p.$$

This decrypts to $m + m'$, provided $m + m' < p$.

The product modulo pq of two ciphertexts, $c = m + rp$ and $c' = m' + r'p$, is

$$\mathsf{Mult}(c, c') = cc' \mod pq$$
$$= mm' + (rm' + r'm + rr'p)p,$$

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

Security of the system is provided by the *partial approximate common divisor problem* (PACDP), first posed by Howgrave-Graham [31], but can be formulated [14, 16] as:

Definition 1. *(Partial approximate common divisor problem.) Suppose we are given one input $x_0 = pr_0$ and n inputs $x_i = pr_i + m_i$, $i \in [1, n]$. We have a bound B such that $|m_i| < B$ for all i. Under what conditions on the variables, r_i and m_i, and the bound B, can an algorithm be found that can uniquely determine p in a time which is 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 ρ, 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 $(x_i \mod p)$ for $i \in [2, n]$. As discussed in section 1.3, the search will requires 2^ρ gcd operations in expectation.

Several attempts have been made to solve the PACDP [14, 16, 31], resulting in theoretically faster algorithms for some cases of the problem. However, our parameters for p and q are chosen to negate the attacks of [16, 31]. The paper [14] gives an algorithm requiring only $\sqrt{M}$ polynomial time operations in the special case that $\mathcal{D}$ is the uniform distribution on $[0, M)$, and hence $\rho = \lg M$. No algorithm running in time subexponential in ρ is known for this problem in the worst case. Therefore, if ρ is large enough, the encryption should be secure.

In actuality, our system is a special case of PACDP because we use the residues of the approximate prime multiples modulo a distinct semiprime modulus. A semiprime is a natural number that is the product of two prime numbers. A distinct semiprime is a semiprime where the prime factors are distinct. We denote this case of PACDP as the *semiprime partial approximate common divisor problem* (SPACDP). Although it is a restriction, there is no reason to believe that this 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. We will show that breaking HE1 is equivalent to semiprime factorisation. Therefore, our scheme is at least as secure as unpadded RSA [42].

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.*

There is a variant of brute force attack on this system, which we will call a *collision attack*. Suppose we have a pair of equal plaintexts $m_1 = m_2$. Then the difference between their encryptions $(c_1 - c_2)$ is an encryption of 0, and the scheme is subject to KPA. In fact, if we have n plaintexts $m_1, m_2, \ldots, m_n$, and there exist $i, j \in [1, n]$ with $m_i = m_j$, the product $\Pi_{1 \leq i < j \leq n}(c_j - c_i)$ is an encryption of 0. However, if there is sufficient entropy, this attack is not possible.

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

Thus, if we have 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 $2^{-\rho/3}$, smaller than any inverse polynomial in λ. Hence, for large enough λ, collision attack is infeasible.

A similar collision attack can be made against the schemes described below. We will not discuss the details, since they are almost identical to those above.

2.2 Insufficient Entropy (HE1N)

Suppose now that the integer inputs $m_i, i \in [1, n]$, are distributed with entropy ρ, where ρ 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 κ) of an integer κ, chosen so that the entropy $\rho' = \rho + \lg \kappa$ is large enough to negate a brute force guessing attack. We also require $\kappa > (n+1)^d M^d$, so that $P(m_1, m_2, \ldots, m_n) < \kappa$. As a result of the extra linear term in the ciphertext, we compute $P(m_1, \ldots, m_n, \kappa)$ instead. We can easily retrieve $P(m_1, \ldots, m_n)$ from $P(m_1, \ldots, m_n, \kappa)$. KeyGen now chooses 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 $s_1, s_2, \ldots, s_n \in [0, \kappa)$. The secret key, sk, is now (κ, p).

We can set these parameters to practical values. If we assume $M \approx 2^\rho$ and large enough n, as in section 2.1, 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 encrypt a plaintext, m, as

$$\mathsf{Enc}(m, \mathsf{sk}) = m + s\kappa + rp \mod pq,$$

where $r \xleftarrow{\$} [1, q]$ and $s \xleftarrow{\$} [0, \kappa)$. We decrypt a ciphertext, c, as

$$\mathsf{Dec}(c, \mathsf{sk}) = (c \mod p) \mod \kappa.$$

Addition and multiplication of ciphertexts is as above.

The use of random noise gives the encryption the following "indistinguishability" property, which implies that the system satisfies IND-CPA [4, 6].

Theorem 3. *For any encryption c, $c \mod \kappa$ is polynomial time indistinguishable from the uniform distribution on $[0, \kappa)$. Thus HE1 satisfies IND-CPA, under the assuption that SPACDP is not polynomial time solvable.*

3 Adding a dimension

In this section we discuss adding an additional dimension to the ciphertext, which becomes a 2-vector. 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.

3.1 Sufficient entropy (HE2)

As with HE1, it is assumed that the inputs m_i ($i \in [1, n]$) are of sufficient entropy. p and q are chosen by KeyGen according to the bounds given in section 2.1. KeyGen also sets $\boldsymbol{a} = [a_1 \ a_2]^T$, where $a_i \xleftarrow{\$} [1, pq]$ ($i \in [1, 2]$) such that $a_1, a_2, a_1 - a_2 \neq 0 \pmod{p \text{ and mod } q}$. The secret key sk is $(p, \boldsymbol{a})$ and the public parameters are pq and R. R is the re-encryption matrix, which is detailed below.

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 $a_1, a_2 \xleftarrow{\$} [0, pq]$.

Encryption

We encrypt a plaintext integer m as the 2-vector $\boldsymbol{c}$,

$$\boldsymbol{c} = \mathsf{Enc}(m, \mathsf{sk}) = (m + rp)\boldsymbol{1} + s\boldsymbol{a} \mod pq,$$

where $\mathbf{1} = [1\ 1]^T$, $r \xleftarrow{\$} [0, q)$, and $s \xleftarrow{\$} [0, pq)$. We construct $\boldsymbol{c}_\star$, where $c_3 = f(c_1, c_2)$ for a given linear function f. We will use $f(c_1, c_2) = 2c_1 - c_2$, though we only require $f(c_1, c_2) \neq c_1, c_2$. Therefore, $c_3 = (m + rp) + sa_3$ mod pq, for $a_3 = 2a_1 - a_2$.

Theorem 4. *The encryption scheme produces ciphertexts with components which are random integers modulo* pq.

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

Decryption

To decrypt, we eliminate s from $\boldsymbol{c}$ (modulo p), giving

$$\mathsf{Dec}(\boldsymbol{c}, \mathsf{sk}) = \boldsymbol{\gamma}^T \boldsymbol{c} \mod p,$$

where $\boldsymbol{\gamma}^T = (a_2 - a_1)^{-1}[a_2\ \ -a_1]$. We call γ the *decryption vector*.

Addition

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

$$\mathsf{Add}(\boldsymbol{c}, \boldsymbol{c}') = \boldsymbol{c} + \boldsymbol{c}' \mod pq.$$

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

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

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

Multiplication

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

$$\boldsymbol{z}_\star = \boldsymbol{c}_\star \circ \boldsymbol{c}'_\star = \begin{bmatrix} c_1 c_1' \\ c_2 c_2' \\ c_3 c_3' \end{bmatrix} \mod pq$$

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

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

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

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

We achieve this by multiplying $\boldsymbol{z}_\star$ by R, a 2×3 matrix,

$$\begin{bmatrix} 1 - 2\alpha_1 & \alpha_1 & \alpha_1 \\ -2\alpha_2 & \alpha_2 + 1 & \alpha_2 \end{bmatrix},$$

where α_1 and α_2 are parameters to be decided.

It is easy to check that $R\mathbf{1}_\star = \mathbf{1}$ and $R\boldsymbol{a}_\star = \boldsymbol{a}$, independently of a_1, a_2. Now

$$(R\boldsymbol{a}_\star^{\circ 2})_1 = (1 - 2\alpha_1)a_1^2 + \alpha_1 a_2^2 + \alpha_1(2a_1 - a_2)^2$$
$$= a_1^2 + \alpha_1((2a_1 - a_2)^2 + a_2^2 - 2a_1^2)$$
$$= a_1^2 + 2\alpha_1(a_2 - a_1)^2$$
$$(R\boldsymbol{a}_\star^{\circ 2})_2 = -2\alpha_2 a_1^2 + (\alpha_2 + 1)a_2^2 + \alpha_2(2a_1 - a_2)^2$$
$$= a_2^2 + \alpha_2((2a_1 - a_2)^2 + a_2^2 - 2a_1^2)$$
$$= a_2^2 + 2\alpha_2(a_2 - a_1)^2$$

9

Let $\beta = 2(a_2 - a_1)^2$. Thus, $\beta^{-1} \mod pq$ exists. Therefore, if we set

$$\alpha_1 = \beta^{-1}(\sigma a_1 + \varrho p - a_1^2), \qquad \alpha_2 = \beta^{-1}(\sigma a_2 + \varrho p - a_2^2), \tag{1}$$

where $\varrho \xleftarrow{\$} [0, q]$ and $\sigma \xleftarrow{\$} [0, pq)$, then we obtain the identity

$$R\boldsymbol{a}_\star^{\circ 2} = \varrho p \mathbf{1} + \sigma \boldsymbol{a}.$$

Observe that α_1, α_2 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, and solving them is as hard as semiprime factorisation in the worst case [41].

We re-encrypt by applying R to $\boldsymbol{z}_\star$, i.e. $\boldsymbol{z}' = R\boldsymbol{z}_\star$, so

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

for some integers r_2, s_2. So $\boldsymbol{z}'$ is a valid encryption of mm'.

Therefore, the homomorphic equivalent of a multiplication operation is defined as

$$\mathsf{Mult}(\boldsymbol{c}, \boldsymbol{c}') = \boldsymbol{c} \cdot \boldsymbol{c}' = R(\boldsymbol{c}_\star \circ \boldsymbol{c}'_\star) \quad (\bmod\ pq),$$

where $\cdot$ is a product on $\mathbb{Z}_{pq}^2$ and $\boldsymbol{c}_\star \circ \boldsymbol{c}'_\star$ is the Hadamard product modulo pq of the two extended ciphertext vectors $\boldsymbol{c}_\star$ and $\boldsymbol{c}'_\star$. Thus, the public parameters of the system are the modulus pq and the re-encryption matrix R, i.e. (pq, R).

Observe that, independently of $\boldsymbol{a}$,

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

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

Hardness

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

Theorem 5. *SPACDP is of equivalent complexity to the special case of HE2 where $\delta = a_2 - a_1$ $(0 < \delta < q)$ is known.*

Observe that, without knowing the parameter $k = a_2 - a_1$, HE2 cannot be reduced to SPACDP in this way. Thus HE2 is seemingly more secure than HE1.

Cryptanalysis

Each new ciphertext $\boldsymbol{c}$ 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 $\boldsymbol{c}, \boldsymbol{c}'$, we can determine

$$\begin{aligned}
(c_1 - m) &= rp + sa_1, & (c_2 - m) &= rp + sa_2, \\
(c_1' - m') &= r'p + s'a_1, & (c_2' - m') &= r'p + s'a_2,
\end{aligned}$$

so

$$\begin{aligned}
(c_1 - m)(c_2' - m') &- (c_2 - m)(c_1' - m') \\
&= (a_2 - a_1)(rs' - r's)p \quad (\bmod\ pq)
\end{aligned}$$

10

Since $a_2 \neq a_1$, and $sr' \neq s'r$ with high probability, this is a nonzero multiple of p, νp say. We may assume $\nu < q$, so $p = \gcd(\nu p, pq)$. We can now solve the linear system $\gamma^T[\boldsymbol{c} \ \boldsymbol{c}'] = [m \ m'] \mod p$ 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 2ρ, so we have effectively squared the number of guesses needed to break the system relative to HE1. So HE2 can tolerate somewhat smaller entropy than HE1. We note further that HE2 does not seem immediately vulnerable to other attacks on HE1 [14, 16, 31].

3.2 Insufficient entropy (HE2N)

In this section we extend HE1N above (section 2.2) to two dimensions. KeyGen chooses p, q and κ according to the bounds given in section 2.2 and $\boldsymbol{1}$, $\boldsymbol{a}$ are as in section 3.1. The secret key is $(\kappa, p, \boldsymbol{a})$, and the public parameters are pq and R, as defined in section 3.1.

We encrypt a plaintext integer $m \in [0, M)$ as a 2-vector $\boldsymbol{c}$,

$$\mathsf{Enc}(m, \mathsf{sk}) = \boldsymbol{c} = (m + rp + s\kappa)\boldsymbol{1} + t\boldsymbol{a} \mod pq,$$

where r is as in section 3.1, $s \stackrel{\$}{\leftarrow} [0, \kappa)$, and $t \stackrel{\$}{\leftarrow} [0, pq)$.

We decrypt a ciphertext $\boldsymbol{c}$ by

$$\mathsf{Dec}(\boldsymbol{c}, \mathsf{sk}) = (\boldsymbol{\gamma}^T \boldsymbol{c} \mod p) \mod \kappa,$$

where $\boldsymbol{\gamma}^T$ is defined as in 3.1.

Addition and multiplication of ciphertexts are defined as in section 3.1.

Finally, we note that HE2N satisfies Theorem 3.

4 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 $k = 2$.

4.1 Sufficient entropy (HEk)

We now generalise HE2 to k dimensions. KeyGen, randomly chooses p and q according to the bounds given in section 3.1. KeyGen sets $\boldsymbol{a}_j \stackrel{\$}{\leftarrow} [1, pq)^k$, $\forall j \in [1, k-1]$. The secret key, sk, is $(p, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1})$, and the public parameters are pq and R. Again, R is detailed below.

With regard to computational overhead, 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.

Encryption

A plaintext, $m \in [0, M]$, is enciphered as

$$\mathsf{Enc}(m, \mathsf{sk}) = \boldsymbol{c} = (m + rp)\boldsymbol{1} + \sum_{j=1}^{k-1} s_j \boldsymbol{a}_j \mod pq$$

where $\boldsymbol{c}$ is a k-vector, $r \stackrel{\$}{\leftarrow} [0, q)$, and $\forall j, s_j \stackrel{\$}{\leftarrow} [0, pq)$. Let $\boldsymbol{a}_0 = \boldsymbol{1}$, and $A_k = [\boldsymbol{a}_0 \ \boldsymbol{a}_1 \ \ldots \ \boldsymbol{a}_{k-1}]$. We wish the columns of A_k to form a basis for $\mathbb{Z}_{pq}^k$. We will show that they do so with high probability. In the unlikely event that they do not, we generate new vectors until they do.

Lemma 6. $\Pr(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1} \text{ do not form a basis}) \leq (k-1)(1/p + 1/q)$.

We extend our definition of an augmented vector, $\boldsymbol{v}_\star$, for a k-vector, $\boldsymbol{v}$, such that $\boldsymbol{v}_\star$ is a $\binom{k+1}{2}$-vector, with components v_i $(1 \le i \le k)$ followed by $2v_i - v_j$ $(1 \le i < j \le k)$. In general, for $\ell > k$, $v_\ell = 2v_i - v_j$, where $\ell = \binom{i}{2} + k + j - 1$. Note that $\boldsymbol{v}_\star = U_k \boldsymbol{v}$ 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. Note that $\boldsymbol{v}_\star = U_k \boldsymbol{v}$ implies that $\mathbf{1}_\star$ is the $\binom{k+1}{2}$ vector of 1's, and that $*$ is a linear mapping, i.e. $(r_1 \boldsymbol{v}_1 + r_2 \boldsymbol{v}_2)_\star = r_1 \boldsymbol{v}_{1*} + r_2 \boldsymbol{v}_{2*}$.

Decryption

$$\mathsf{Dec}(\boldsymbol{c}, \mathsf{sk}) = \boldsymbol{\gamma}^T \boldsymbol{c} \mod p.$$

where $\boldsymbol{\gamma}^T = (A_k^{-1})_1$ is the first row of A_k^{-1}. We call $\boldsymbol{\gamma}$ the *decryption vector*, as in HE2.

Addition

Addition of ciphertexts is the vector sum of the ciphertext vectors as with HE2.

Multiplication

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

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

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

The re-encryption matrix, R, is $k \times \binom{k+1}{2}$. We require that $R\boldsymbol{v}_\star = \boldsymbol{v}$, for all $\boldsymbol{v}$.

Lemma 7. *Let $A_{\star k} = [\boldsymbol{a}_{\star 0}\ \boldsymbol{a}_{\star 1}\ \ldots\ \boldsymbol{a}_{\star, k-1}]$, where the columns of A_k form a basis for $\mathbb{Z}_{pq}^k$. If $RA_{\star k} = A_k$, then $R\boldsymbol{v}_\star = \boldsymbol{v}$ for all $\boldsymbol{v} \in \mathbb{Z}_{pq}^k$.*

The condition $RA_{\star k} = A_k$ can be written more simply, since it is $RU_k A_k = A_k$. Postmultiplying by A_k^{-1} gives $RU_k = I_k$.

Since $RA_{\star k} = A_k$, we have

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

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

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

$$R(\boldsymbol{a}_{\star i} \circ \boldsymbol{a}_{\star j}) = \varrho_{ij} p \mathbf{1} + \sum_{l=1}^{k-1} \sigma_{ijl} \boldsymbol{a}_l \tag{2}$$

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

$$RA_{\star k}^{\circ 2} = [A_k \mid C_k]. \tag{3}$$

12

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

The system of equations (3) has a solution provided that $A_{\star k}^{\circ 2}$ has an inverse mod pq. We prove that this is true with high probability. Again, in the unlikely event that this is not true, we generate new vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1}$ until it is.

Theorem 8. $A_{\star k}^{\circ 2}$ *has no inverse* $\bmod pq$ *with probability at most* $(k^2 - 1)(1/p + 1/q)$.

Note that Theorem 8 subsumes Lemma 6, since the first k columns of $A_{\star k}^{\circ 2}$ contain A_k as a submatrix, and must be linearly independent.

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

Cryptanalysis

Note that p can still be determined if we know m_i for k ciphertexts. Then let

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

and let

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

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

where r_i, s_{ij} refer to $\boldsymbol{c}_i$. Then $C = A_k W$, 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 $\gcd(\det C, pq)$. Then p can be recovered if $\det C \neq 0$.

Lemma 9. $\Pr(\det C = 0 \bmod pq) \leq (2k - 1)(1/p + 1/q)$.

Once we have recovered p, we can use the known m_i to determine the decryption vector $\boldsymbol{\gamma}$, by solving linear equations. Let

$$C_0 = [\boldsymbol{c}_1 \ \boldsymbol{c}_2 \ \ldots \ \boldsymbol{c}_k], \quad \boldsymbol{m}^T = [m_1 \ m_2 \ \ldots \ m_k].$$

Lemma 10. $\Pr(\det C_0 = 0 \bmod pq) \leq (2k - 1)(1/p + 1/q)$.

Thus, with high probability, we can solve the system

$$\boldsymbol{\gamma}^T C_0 = \boldsymbol{m}^T \qquad \bmod p$$

uniquely, to recover $\boldsymbol{\gamma}$ and enable decryption of an arbitrary ciphertext. However, encryption of messages is not possible, since we gain little information about $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k$. 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 μ^k guesses, where μ is the number of guesses needed to break HE1. So HEk can tolerate much smaller entropy than HE1, provided k is large enough. If k is sufficiently large, the scheme appears secure without adding noise, though it does not have the other advantages of adding noise. We discuss this further in section 6.

Fixing an insecurity for $k > 2$

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

The equations

$$R(\boldsymbol{a}_{\star i} \circ \boldsymbol{a}_{\star j}) \;=\; p\varrho_{ij}\,\mathbf{1} + \sum_{l=1}^{k-1} \sigma_{ijl}\,\boldsymbol{a}_l, \tag{4}$$

define a product $\cdot$ on $\mathbb{Z}_{pq}^k$ so that $\boldsymbol{c} \cdot \boldsymbol{c}' = R(\boldsymbol{c}_\star \circ \boldsymbol{c}'_\star)$. 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 $\mathbf{1}$ [44]. 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 not be associative, i.e. we can have

$$R(R(\boldsymbol{c}_{1\star} \circ \boldsymbol{c}_{2\star})_\star \circ \boldsymbol{c}_{3\star}) = (\boldsymbol{c}_1 \cdot \boldsymbol{c}_2) \cdot \boldsymbol{c}_3$$
$$\neq \boldsymbol{c}_1 \cdot (\boldsymbol{c}_2 \cdot \boldsymbol{c}_3) = R(\boldsymbol{c}_{1\star} \circ R(\boldsymbol{c}_{2\star} \circ \boldsymbol{c}_{3\star})_\star).$$

This leads to the following potential insecurity. We must have

$$\boldsymbol{\gamma}^T((\boldsymbol{c}_1 \cdot \boldsymbol{c}_2) \cdot \boldsymbol{c}_3) \;=\; \boldsymbol{\gamma}^T(\boldsymbol{c}_1 \cdot (\boldsymbol{c}_2 \cdot \boldsymbol{c}_3)) \qquad (\mathrm{mod}\ p), \tag{5}$$

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

$$[\boldsymbol{c}_i, \boldsymbol{c}_j, \boldsymbol{c}_l] \;= \boldsymbol{c}_i \cdot (\boldsymbol{c}_j \cdot \boldsymbol{c}_l) - (\boldsymbol{c}_i \cdot \boldsymbol{c}_j) \cdot \boldsymbol{c}_l$$
$$= rp\mathbf{1} + \sum_{l=1}^{k-1} s_l\,\boldsymbol{c}_l \qquad (\mathrm{mod}\ pq).$$

Thus $[\boldsymbol{c}_i, \boldsymbol{c}_j, \boldsymbol{c}_l]$ is an encryption of 0. If we can find k such associators from $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n$ which violate (5), then with high probability we will have k linearly independent associators. We can use use these to make a collision attack on HEk, in a similar way to that described in section 2.1. We use the gcd method to determine p, and then $\boldsymbol{\gamma}$, as described in section 4.1. In fact all we need is that (5) holds for any associator. That is, for all $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3$, we need

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

or, equivalently, using the CRT,

$$\boldsymbol{\gamma}^T((\boldsymbol{c}_1 \cdot \boldsymbol{c}_2) \cdot \boldsymbol{c}_3) \;=\; \boldsymbol{\gamma}^T(\boldsymbol{c}_1 \cdot (\boldsymbol{c}_2 \cdot \boldsymbol{c}_3)) \qquad (\mathrm{mod}\ q). \tag{6}$$

By linearity, it follows that (6) 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

$$\boldsymbol{\gamma}^T(\boldsymbol{a}_i \cdot (\boldsymbol{a}_j \cdot \boldsymbol{a}_l)) \;=\; \boldsymbol{\gamma}^T((\boldsymbol{a}_i \cdot \boldsymbol{a}_j) \cdot \boldsymbol{a}_l) \qquad (\mathrm{mod}\ q). \tag{7}$$

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

$$[\boldsymbol{a}_i, \boldsymbol{a}_j, \boldsymbol{a}_l] \;= \boldsymbol{a}_i \cdot (\boldsymbol{a}_j \cdot \boldsymbol{a}_l) - (\boldsymbol{a}_i \cdot \boldsymbol{a}_j) \cdot \boldsymbol{a}_l$$
$$= rp\mathbf{1} + \sum_{l=1}^{k-1} s_l\,\boldsymbol{a}_l \qquad (\mathrm{mod}\ pq),$$

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

If $\mathcal{A}_k$ is associative, the problem does not arise, since (7) will be satisfied automatically. Associativity holds for $k \leq 2$, since all we have to check is that $\boldsymbol{a} \cdot (\boldsymbol{a} \cdot \boldsymbol{a}) = (\boldsymbol{a} \cdot \boldsymbol{a}) \cdot \boldsymbol{a}$, which is true by commutativity. Thus HEk with $k \leq 2$ cannot be attacked in this way.

Requiring associativity in $\mathcal{A}_k$ would overconstrain the system, since it imposes $k\binom{k+1}{2}$ equations on the $k\binom{k+1}{2}$ structure constants. We have only $k(k-1)$ undetermined parameters, so this is too much. But all we need is that (7) holds. We have the following.

Lemma 11. (7) *holds if and only if*

$$\sum_{t=1}^{k-1} \sigma_{jlt} \varrho_{it} = \sum_{t=1}^{k-1} \sigma_{ijt} \varrho_{lt} \pmod{q}, \ \forall i, j, l \in [1, k-1].$$

There are several ways to ensure that (7) holds. We will do so by giving the ϱ_{ij} a multiplicative structure.

Lemma 12. *Let* $\tau, \varrho_i \xleftarrow{\$} [0, q)$ ($i \in [1, k-1]$), *let* $\varrho_{ij} = \varrho_i \varrho_j \mod q$, *and let the* σ_{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* $i, j, \ell \in [1, k-1]$, $\boldsymbol{\gamma}^T(\boldsymbol{a}_i \cdot (\boldsymbol{a}_j \cdot \boldsymbol{a}_l)) = \tau \varrho_i \varrho_j \varrho_l \mod q$, *the symmetry of which implies that* (7) *holds.*

Thus the conditions of Lemma 12 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 $\Omega(k^2)$.

4.2 Insufficient entropy (HEkN)

In this section, we generalise HE2N to k dimensions. KeyGen, randomly chooses κ, p and q according to the bounds given in section 3.2. $\forall j$, KeyGen sets $\boldsymbol{a}_j$ as in 4.1. The secret key, sk, is $(\kappa, p, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1})$, and the public parameters are pq and R. R is as given in section 4.1. Note that, as a result of adding the "noise" term, defence against non-associativity is not required.

A plaintext, $m \in [0, M]$, is enciphered as

$$\mathsf{Enc}(m, \mathsf{sk}) = \boldsymbol{c} = (m + rp + s\kappa)\boldsymbol{1} + \sum_{j=1}^{k-1} t_j \boldsymbol{a}_j \pmod{pq}$$

where r, s are as in section 3.2, and $t_j \xleftarrow{\$} [0, pq) \ \forall j \in [1, k)$.

A ciphertext is deciphered by,

$$\mathsf{Dec}(\boldsymbol{c}, \mathsf{sk}) = (\boldsymbol{\gamma}^T \boldsymbol{c} \mod p) \mod \kappa.$$

where $\boldsymbol{\gamma}^T$ is defined as in section 4.1.

Addition and multiplication of ciphertexts are as in section 4.1.

The effective entropy of HEkN 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 HEkN also inherits the conclusions of Theorem 3.

5 An extension of HE2N using the CRT (HE2NCRT)

As an interesting aside, we extend HE2N (section 3.2) using a technique inspired by 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 processors. Each processor computes arithmetic on ciphertexts modulo $p_j q_j$, where p_j, q_j are suitable primes. Also, each processor only receives the jth ciphertext vector of an integer. Addition and multiplication of ciphertexts is as defined in section 3.1, 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 3.2, by not distributing the modulus pq to each processor. Instead, we distribute $p_j q_j$ to the jth processor, for $j \in [1, K]$. This allows us to partition the computation into subcomputations, encrypted using different parameters. Thus, should an attacker compromise one subcomputation, they may gain no knowledge of other subcomputations.

Key Generation

KeyGen, randomly chooses κ as in section 2.2. 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 2.1. Finally, it sets $\boldsymbol{a}_j = [a_{j1} \ a_{j2}]^T$, where $a_{jk} \xleftarrow{\$} [1, p_j q_j)$ ($j \in [1, K], k \in [1, 2]$) such that $a_{j1} \neq a_{j2} \mod p$ and $a_{j1} \neq a_{j2} \mod q$. The secret key, sk, is $(\kappa, p_1, \ldots, p_K, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_K)$, and the public parameters are $p_j q_j$ ($j \in [1, K]$) and R_j ($j \in [1, K]$), where each R_j is defined as R in section 3.1.

Encryption

We encrypt an integer, m_i ($i \in [1, n]$), as the set of K 2-vectors, $\boldsymbol{c}_{ij}$,

$$\boldsymbol{c}_{ij} = (m_i + r_{ij} p_j + s_i \kappa) \mathbf{1} + t_{ij} \boldsymbol{a}_j \mod p_j q_j \ (j \in [1, K]),$$

where $r_{ij} \xleftarrow{\$} [0, q_j)$, $s_i \xleftarrow{\$} [0, \kappa)$, and $t_{ij} \xleftarrow{\$} [0, p_j q_j)$ ($i \in [1, n], j \in [1, K]$).

Decryption

To decrypt, we first decrypt the jth ciphertext of the computational result $\boldsymbol{c}_j$ as in section 3.1, to give

$$P_j = \boldsymbol{\gamma}_j^T \boldsymbol{c}_j \mod p_j,$$

where P_j is the residue of $P(m_1, m_2, \ldots, m_n, \kappa) \mod p_j$ and $\boldsymbol{\gamma}_j^T = (a_{j2} - a_{j1})^{-1} [a_{j2} \ -a_{j1}]$.

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

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

where $M_j = \Pi / p_j$ and $\mu_j = M_j^{-1} \mod p_j$.

Addition and Multiplication

Addition of ciphertexts is performed as in 3.1. Multiplication of ciphertexts on processor j is now

$$\mathsf{Mult}(c_j, c_j') = R_j (c_{j\star} \circ c_{j\star}').$$

Clearly HEkN could be extended to HEkNCRT in a similar way, but we will not discuss the details here.

6 Fully Homomorphic System

We return to HEk, presented above in section 4.1. We will show that, for large enough k, this can be used to create an FHE system.

We may use HEk to evaluate an arithmetic circuit homomorphically, where $\mathsf{R} = \mathbb{Z}_{pq}$. However, this system is only somewhat homomorphic. If the computational result grows larger than p, we are unable to successfully decrypt it. This restricts us to arithmetic circuits of bounded depth to avoid the blow up. To make it fully homomorphic, we consider Boolean circuits.

Typically, we will use the binary Boolean functions, AND, OR, and NOT in the Boolean circuit. However, we may use fewer functions. Any Boolean circuit may be represented using only NAND gates [45]. Recall that the indegree of any gate in the circuit is always 2, but the outdegree is arbitrary. 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 by $O \subseteq V$. The inputs have indegree 0, and the outputs have outdegree 0, but we regard the

inputs as having indegree 1, and the outputs as having outdegree 1, with wires from and to the external environment Λ.

Note that constant input bits can easily be eliminated from the circuit, so we assume there are none, to avoid an obvious KPA. Even so, if we represent the bit values $0, 1$ by encrypting known constants α_0, α_1, the HEk system is open to a simple KPA. For any ciphertext $\boldsymbol{c}$, we can compute $\boldsymbol{c}' = (\boldsymbol{c} - \alpha_0 \mathbf{1}) \cdot (\boldsymbol{c} - \alpha_1 \mathbf{1})$. Then $\boldsymbol{c}'$ is an encryption of 0. By repeating this on k ciphertexts, we can obtain k linearly independent encryptions of zero with high probability. Once we have done this, we can determine p and $\boldsymbol{\gamma}$ as in section 4.1. The problem, of course, is that we have not increased the entropy of the input data.

Therefore, we must add noise to the ciphertexts, but we will do this so as to ensure that the noise does not grow with the the depth of the circuit. 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 $\omega_{0e} = 2s_{0,e}$, $\omega_{1e} = 1 + 2s_{1,e}$, where $s_{0,e}, s_{0,e} \xleftarrow{\$} [0, \kappa)$. Thus $b_e = w_e \bmod 2$, and the noise has entropy $\lg \kappa$. The value of κ is chosen as large as possible such that we can correctly evaluate any polynomial of degree 2 in two variables. For each input $i \in I$, we represent the input bit value b_i similarly, by $x_i \in \{\omega_{0i}, \omega_{1i}\}$. The inputs and the wires in the circuit are encrypted using HEk, the inputs directly and the other wires indirectly as described below. As discussed in section 4.1, we need k known plaintexts to break HEk. The plaintexts are the encrypted bits $w_e \bmod 2$, so a brute force attack requires guessing at least 2^k bits. So, by setting $k \gg \log \lambda$, a brute force attack on the system requires time superpolynomial in the security parameter λ.

An input $i \in I$ has a wire (Λ, 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 $w \in \{\gamma_0, \gamma_1\}$. (We suppress the wire labels e when they are clear from the context.) This requires

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

The encrypted coefficients of this function are stored as data for the wire e. Note that all computations are mod pq, and the required inverses exist because the numbers involved are less than κ.

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 $w \in \{\gamma_0, \gamma_1\}$. This requires

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

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

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

Thus we replace 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" [5, 25], 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 ω_{0e}, ω_{1e} for w_e. Suppose we can observe the encrypted value on wire e three times giving cyphertexts $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3$. Two of these are encryptions of the same value $2s_{0,e}$ or $1 + 2s_{1,e}$. Thus $(\boldsymbol{c}_1 - \boldsymbol{c}_2) \cdot (\boldsymbol{c}_1 - \boldsymbol{c}_3) \cdot (\boldsymbol{c}_2 - \boldsymbol{c}_3)$ 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 2.

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.

7 Experimental Results

HE1, HE1N, HE2, and HE2N have been implemented in pure unoptimised Java using the JScience mathematics library [19]. Secure pseudo-random numbers are generated using the ISAAC algorithm [33]. The

Alg.	Parameters			Encryption			MR Job		Decrypt(ms)
	d	ρ	ρ'	Init(s)	Enc(μs)	Exec(s)	Prod(μs)	Sum(μs)	
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

Table 1: Timings for each experimental configuration. *Init* is the initialisation time for the encryption algorithm, *Enc* is the mean time to encrypt a single integer, *Exec* is the 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.

ISAAC cipher is seeded using the Linux `/dev/random` source. This prevents the weakness in ISAAC shown by Aumasson [1].

We devised a simple evaluation experiment to generate a fixed (24,000) number of encrypted inputs and then perform a homomorphic inner product on those inputs using a Hadoop MapReduce (MR) algorithm. On the secure client side, the MR input is generated as pseudo-random ρ-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. 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 these products. The MR algorithm divides the input file so that each mapper receives an equal number of lines of input, thereby 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 (ρ), and effective entropy of inputs after adding "noise" (ρ', for the 'N' variant algorithms only), was executed 10 times. The mean results are tabulated in Table 1.

Our results compare extremely favourably with Table 2 of [36]. 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, to decrypt our results we take the modulus modulo a large primes rather than 2 as in the case of [36]. This is obviously less efficient. For homomorphic sums and products, our algorithms perform approximately 100 times faster. [36] only provides experimental data for computing degree 2 polynomials. We have provided experimental results for the computation of higher degree polynomials.

Similarly, compared with figure 13 of [40], 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 figure 12, it is unclear how many rows exist in their test database. Nevertheless, our results for computing homomorphic sums compare favourably with those given. It should be noted that CryptDB [40] only supports homomorphic sums and is incapable of computing an inner product. Therefore, we only compare the homomorphic sum timings.

Table 1 of [46] is unclear on whether the values are aggregate timings or the timing per operation. Even assuming that they are aggregate values, our results are approximately 100 times faster than those presented for homomorphic sum and product operations. We also note that Crypsis [46] uses two different encryption schemes for integers, ElGamal [23] and Paillier [39], which only support addition or multiplication but not both. No discussion of computation of an inner product is made in [46] 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.

We note that there are some apparent anomalies in the data. JScience implements arbitrary precision integers as an array of Java `long` (64-bit) integers to store the bit representation of an integer. It may be the case that this underlying representation is optimal for some of our test configurations and suboptimal for others, causing unexpected results. Another possibility is that the unexpected results may be as a result of JVM heap increases and garbage collection which may have been more prevalent in certain test configurations.

8   Conclusion

In this paper we have presented several new homomorphic encryption schemes intended for use in a practical SSCC system. 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.

As they are only somewhat homomorphic, each of these schemes has a concern that the computational result will 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 suitably bounded degree.

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 would make the system practical for computing low degree polynomials. Similar schemes [18, 20] produce ciphertexts infeasibly larger than the corresponding plaintext, which is a single bit. For example, it should be noted, that even the practical CryptDB [40], which is not fully 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, and the blow-up may be acceptable. 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.

We have also presented a hierarchy of systems, HEk, with increasing levels of security. These seem to be of practical interest for small $k > 2$, but seem impractical for large k.

Finally, we presented a fully homomorphic scheme based on HEk for large enough k, which seems of purely theoretical interest. The scheme is capable of computing an arbitrary depth Boolean circuit without employing the techniques used in other fully homomorphic systems [10, 13, 24].

We have implemented and evaluated the HE1, HE1N, HE2 and HE2N schemes as part of an SSCC system as discussed in section 7. Our results are extremely favourable when compared with [36, 40, 46]. So much so, that our MapReduce job execution times remain low even when using the largest set of parameters for HE2N. We believe that this demonstrates the suitability of our schemes for the encryption of integers in cloud computations.

References

[1] J.-P. Aumasson. *On the pseudo-random generator ISAAC*. Cryptology ePrint Archive, Report 2006/438. http://eprint.iacr.org/2006/438. 2006.

[2] L. Babai and L. Fortnow. 'Arithmetization: A new method in structural complexity theory'. In: *computational complexity* 1.1 (1991), pp. 41–66. DOI: 10.1007/BF01200057.

[3] M. Bellare and P. Rogaway. *Introduction to Modern Cryptogrpahy*. 2005.

[4] M. Bellare et al. 'A Concrete Security Treatment of Symmetric Encryption'. In: *Proc. FOCS '97*. IEEE, 1997, pp. 394–403. DOI: 10.1109/SFCS.1997.646128.

[5] M. Bellare et al. 'Foundations of Garbled Circuits'. In: *Proc. CCS '12*. ACM, 2012, pp. 784–796. DOI: 10.1145/2382196.2382279.

[6] M. Bellare et al. 'Relations Among Notions of Security for Public-Key Encryption Schemes'. In: *Proc. CRYPTO '98*. Springer, 1998, pp. 26–45. DOI: 10.1007/BFb0055718.

[7] E. R. Berlekamp. 'Factoring Polynomials Over Large Finite Fields'. In: *Mathematics of Computation* 24.111 (1970), pp. 713–735.

[8] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. Draft 0.2, 2015.

[9] D. Boneh et al. 'Private Database Queries Using Somewhat Homomorphic Encryption'. In: *Proc. ACNS '13*. Springer, 2013, pp. 102–118. DOI: 10.1007/978-3-642-38980-1_7.

[10] Z. Brakerski. 'Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP'. In: *Proc. CRYPTO '12*. Springer, 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50.

[11] Z. Brakerski and V. Vaikuntanathan. 'Efficient Fully Homomorphic Encryption from (Standard) LWE'. In: *Proc. FOCS '11*. IEEE, 2011, pp. 97–106. DOI: 10.1109/FOCS.2011.12.

[12] Z. Brakerski and V. Vaikuntanathan. 'Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages'. In: *Proc. CRYPTO 2011*. Springer, 2011, pp. 505–524. DOI: 10.1007/978-3-642-22792-9_29

[13] Z. Brakerski et al. '(Leveled) Fully Homomorphic Encryption Without Bootstrapping'. In: *Proc. ITCS '12*. ACM, 2012, pp. 309–325. DOI: 10.1145/2090236.2090262.

[14] Y. Chen and P. Q. Nguyen. 'Faster Algorithms for Approximate Common Divisors: Breaking Fully Homomorphic Encryption Challenges over the Integers'. In: *Proc. EUROCRYPT '12*. Springer, 2012, pp. 502–519. DOI: 10.1007/978-3-642-29011-4_30.

[15]   J. H. Cheon et al. 'CRT-Based Fully Homomorphic Encryption Over the Integers'. In: *Information Sciences* 310 (2015), pp. 149–162. DOI: 10.1016/j.ins.2015.03.019.

[16]   H. Cohn and N. Heninger. 'Approximate common divisors via lattices'. In: *Proc. ANTS-X*. Vol. 1. Mathematical Sciences Publishers, 2012, pp. 271–293. DOI: 10.2140/obs.2013.1.271.

[17]   D. Coppersmith. 'Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities'. In: *J. Cryptology* 10.4 (1997), pp. 233–260. DOI: 10.1007/s001459900030.

[18]   J.-S. Coron et al. 'Fully Homomorphic Encryption over the Integers with Shorter Public Keys'. In: *Proc. CRY-PTO '11*. Springer, 2011, pp. 487–504. DOI: 10.1007/978-3-642-22792-9_28.

[19]   J.-M. Dautelle. *JScience*. Sept. 2014. URL: http://jscience.org.

[20]   M. van Dijk et al. 'Fully Homomorphic Encryption over the Integers'. In: *Proc. EUROCRYPT '10*. Springer, 2010, pp. 24–43. DOI: 10.1007/978-3-642-13190-5_2.

[21]   M. van Dijk and A. Juels. 'On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Co-mputing'. In: *Proc. HotSec '10*. USENIX, 2010, pp. 1–8.

[22]   L. Ducas and D. Micciancio. 'FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second'. In: *Proc. EUROCRYPT '15*. Springer, 2015, pp. 617–640. DOI: 10.1007/978-3-662-46800-5_24.

[23]   T. ElGamal. 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms'. In: *IEEE Trans. Inf. Theory* 31.4 (1985), pp. 469–472. DOI: 10.1109/TIT.1985.1057074.

[24]   C. Gentry. 'Fully Homomorphic Encryption Using Ideal Lattices'. In: *Proc. STOC '09*. ACM. 2009, pp. 169–178.

[25]   O. Goldreich et al. 'How to Play ANY Mental Game'. In: *Proc. STOC '87*. ACM, 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[26]   S. Goldwasser and S. Micali. 'Probabilistic Encryption'. In: *J. Comput. Syst. Sci.* 28.2 (1984), pp. 270–299.

[27]   S. Goldwasser et al. 'How to Run Turing Machines on Encrypted Data'. In: *Proc. CRYPTO '13*. Vol. 2. Springer, 2013, pp. 536–553. DOI: 10.1007/978-3-642-40084-1_30.

[28]   S. Goldwasser et al. 'Reusable Garbled Circuits and Succinct Functional Encryption'. In: *Proc. STOC '13*. ACM, 2013, pp. 555–564. DOI: 10.1145/2488608.2488678.

[29]   S. Halevi and V. Shoup. 'Bootstrapping for HElib'. In: *Proc. EUROCRYPT '15*. Springer, 2015, pp. 641–670. DOI: 10.1007/978-3-662-46800-5_25.

[30]   S. Halevi and V. Shoup. *HELib*. URL: https://github.com/shaih/HElib.

[31]   N. Howgrave-Graham. 'Approximate Integer Common Divisors'. In: *Cryptography and Lattices*. Springer, 2001, pp. 51–66. DOI: 10.1007/3-540-44670-2_6.

[32]   P. Hrubeš and A. Yehudayoff. 'Arithmetic Complexity in Algebraic Extensions'. In: *Theory of Computing* 7 (2011), pp. 119–129.

[33]   B. Jenkins. *ISAAC: a fast cryptographic random number generator*. 1996. URL: http://burtleburtle.net/bob/rand/i

[34]   A. Kipnis and E. Hibshoosh. *Efficient Methods for Practical Fully Homomorphic Symmetric-Key Encryption, Randomization and Verification*. Cryptology ePr-int Archive. 2012. URL: https://eprint.iacr.org/2012/63

[35]   T. Kleinjung et al. 'Factorization of a 768-bit RSA Modulus'. In: *Proc. CRYPTO '10*. Springer, 2010, pp. 333–350. DOI: 10.1007/978-3-642-14623-7_18.

[36]   K. Lauter et al. 'Can Homomorphic Encryption Be Practical?' In: *Proc. CCSW '11*. CCSW '11. ACM, 2011, pp. 113–124. DOI: 10.1145/2046660.2046682.

[37]   J. L. Massey. 'Guessing and Entropy'. In: *Proc. ISIT '94*. 1994, p. 204. DOI: 10.1109/ISIT.1994.394764.

[38]   D. Moshkovitz. 'An Alternative Proof of the Schwartz-Zippel Lemma.' In: *Electronic Colloquium on Computational Complexity (ECCC)*. 2010, p. 96.

[39]   P. Paillier. 'Public-Key Cryptosystems Based on Composite Degree Residuosity Classes'. In: *Proc. EUROCRYPT '99*. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[40] R. A. Popa et al. 'CryptDB: Protecting Confidentiality With Encrypted Query Processing'. In: *Proc. SOSP '11*. ACM, 2011, pp. 85–100.

[41] M. O. Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*. Tech. rep. MIT/LCS/TR-212. MIT, 1979, p. 12.

[42] R. L. Rivest et al. 'A Method for Obtaining Digital Signatures and Public-key Cryptosystems'. In: *Commun. ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[43] R. L. Rivest et al. 'On Data Banks and Privacy Homomorphisms'. In: *Foundations of Secure Computation* 4.11 (1978), pp. 169–180.

[44] R. D. Schafer. *An Introduction to Nonassociative Algebras*. Vol. 22. Courier Corporation, 1966.

[45] T. W. Scharle. 'Axiomatization of Propositional Calculus With Sheffer Functors'. In: *Notre Dame J. Formal Logic* 6.3 (1965), pp. 209–217. DOI: 10.1305/ndjfl/1093958259.

[46] J. J. Stephen et al. 'Practical Confidentiality Preserving Big Data Analysis'. In: *Proc. HotCloud '14*. USENIX. 2014, p. 10.

[47] S. D. Tetali et al. 'MRCrypt: Static Analysis for Secure Cloud Computations'. In: *Proc. OOPSLA '13*. ACM. 2013, pp. 271–286.

[48] D. Vizár and S. Vaudenay. 'Cryptanalysis of Chosen Symmetric Homomorphic Schemes'. In: *Stud. Sci. Math. Hung.* 52.2 (2015), pp. 288–306. DOI: 10.1556/012.2015.52.2.1311.

[49] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer, 1999.

[50] L. Xiao et al. *An Efficient Homomorphic Encryption Protocol for Multi-User Systems*. Cryptology ePrint Archive. 2012. URL: https://eprint.iacr.org/2012/193.pdf.

A   Proofs

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 $r \xleftarrow{\$} [1, q)$.

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. □

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

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

Theorem 3. *For any encryption c, $c \bmod \kappa$ is polynomial time indistinguishable from the uniform distribution on $[0, \kappa)$. Thus HE1 satisfies IND-CPA, under the assuption that SPACDP is not polynomial time solvable.*

Proof.

$$c = m + s\kappa + rp = m + rp \mod \kappa,$$

where $r \xleftarrow{\$} [1, q)$. Thus, 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

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

This is exponentially small in the security parameter λ of the system, so the distribution of $c \bmod \kappa$ cannot be distinguished in polynomial time from the uniform distribution. Note further that $c_1 \bmod \kappa$, $c_2 \bmod \kappa$ are independent for any two ciphertexts $c_i = m_i + s_i\kappa + r_ip$ ($i = 1, 2$), since r_1, r_2 are independent.

To show IND-CPA, suppose now that known plaintexts $\mu_1, \ldots, \mu_n$ are encrypted by an oracle for HE1N, giving ciphertexts $c_1, \ldots, c_n$. Then, for $r_i \xleftarrow{\$} [0, q)$, $s_i \xleftarrow{\$} [0, \kappa)$, 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. However, the offsets in this SPACDP are all of the form $\mu_i + s_i\kappa$, for known m_i, and we must make sure this does not provide information about p. To show this, we rewrite the SPACDP as

$$c_i = \mu_i + s_i\kappa + r_ip = \mu_i' + s_i'\kappa, \quad (i = 1, 2, \ldots, n), \tag{8}$$

where $s_i' = s_i + \lfloor (m_i + r_ip)/\kappa \rfloor$, and $\mu_i' = \mu_i + r_ip \pmod{\kappa}$. Now we may view (8) as an ACDP, with "encryptions" μ_i' of the μ_i, and approximate divisor κ. Since ACDP is at least as hard as SPACDP, and the offsets μ_i' are polynomial time indistinguishable from uniform $[0, \kappa)$, from above, we will not be able to determine κ 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. □

Theorem 4. *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 $r \xleftarrow{\$} [0, q)$, $s \xleftarrow{\$} [0, pq)$.

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

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

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

Theorem 5. *SPACDP is of equivalent complexity to the special case of HE2 where $\delta = a_2 - a_1$ $(0 < \delta < q)$ is known.*

Proof. Suppose we have a system of n approximate prime multiples, $m_i + r_i p$ $(i = 1, 2, \ldots, n)$. Then we generate values $a, s_1, s_2, \ldots, s_n \xleftarrow{\$} [0, pq)$, 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 ϱ and σ. We then generate the ciphertexts $\boldsymbol{c}_i$ $(i = 1, 2, \ldots, n)$:

$$\begin{bmatrix} c_{i1} \\ c_{i2} \end{bmatrix} = \begin{bmatrix} m_i + r_i p + s_i a \\ m_i + r_i p + s_i(a + \delta) \end{bmatrix} \pmod{pq}. \tag{9}$$

Thus $c_{i1} - s_i a = c_{i2} - s_i(a + \delta) = m_i + r_i p$. Thus finding the m_i in (9) 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 (9). so $s_i = \delta^{-1}(c_{i2} - c_{i1})$. Since $0 < \delta < q < p$, δ 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 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 $\sigma = 2\delta^2(\alpha_2 - \alpha_1) - 2ka - \delta^2$. 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 [7]. So the case $\boldsymbol{a} = [a \;\; a + \delta]^T$, with known δ, can be attacked using SPACDP on the system

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

$\square$

Lemma 6. $\Pr(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1}$ *do not form a basis*$) \leq (k-1)(1/p + 1/q)$.

Proof. The $\boldsymbol{a}$'s are a basis if A_k^{-1} exists, since then $\boldsymbol{v} = A_k \boldsymbol{r}$ when $\boldsymbol{r} = A_k^{-1}\boldsymbol{v}$, for any $\boldsymbol{v}$. 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} $(0 < i \leq k, 0 < j < k)$, and is not identically zero, since $\det A_k = 1$ if $\boldsymbol{a}_i = \boldsymbol{e}_{i+1}$ $(1 < i < k)$. Also $a_{ij} \xleftarrow{\$} [0, pq)$ implies $a_{ij} \mod p \xleftarrow{\$} [0, p)$ and $a_{ij} \mod q \xleftarrow{\$} [0, q)$. Hence, using the Schwartz-Zippel Lemma (SZL) [38], we have $\Pr(\det A_k = 0 \mod p) \leq (k-1)/p$ and $\Pr(\det A_k = 0 \mod q) \leq (k-1)/q$, and it follows that $\Pr(\nexists (\det A_k)^{-1} \mod pq) \leq (k-1)(1/p + 1/q)$. $\square$ $\square$

Lemma 7. *Let $A_{\star k} = [\boldsymbol{a}_{\star 0} \; \boldsymbol{a}_{\star 1} \; \ldots \; \boldsymbol{a}_{\star, k-1}]$, where the columns of A_k form a basis for $\mathbb{Z}_{pq}^k$. If $RA_{\star k} = A_k$, then $R\boldsymbol{v}_\star = \boldsymbol{v}$ for all $\boldsymbol{v} \in \mathbb{Z}_{pq}^k$.*

Proof. We have $\boldsymbol{v} = A_k \boldsymbol{r}$ for some $\boldsymbol{r} \in \mathbb{Z}_{pq}^k$. Then $A_{\star k} = U_k A_k$ and $\boldsymbol{v}_{\star k} = U_k \boldsymbol{v}$, so $R \boldsymbol{v}_\star = R U_k \boldsymbol{v} = R U_k A_k \boldsymbol{r} = R A_{\star k} \boldsymbol{r} = A_k \boldsymbol{r} = \boldsymbol{v}$. $\qquad\square$ $\qquad\square$

Theorem 8. $A_{\star k}^{\circ 2}$ *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 6. Thus $A_{\star k}^{\circ 2}$ is invertible provided $\det A_{\star k}^{\circ 2} \neq 0 \mod p$ and $\det A_{\star k}^{\circ 2} \neq 0 \mod q$. Let $\boldsymbol{A}$ denote the vector of a_{ij}'s, $(a_{ij} : 1 \leq i \leq k, 1 \leq j < k)$. The elements of $A_{\star k}^{\circ 2}$ are quadratic polynomials over $\boldsymbol{A}$, except for the first column, which has all 1's, and columns $2, 3, \ldots, k$ which are linear polynomials. So $\det A_{\star k}^{\circ 2}$ is a polynomial over $\boldsymbol{A}$ of total degree $2\binom{k}{2} + k - 1 = k^2 - 1$. Thus, unless $\det A_{\star k}^{\circ 2}$ is identically zero as a polynomial over $\boldsymbol{A}$, the SZL [38] implies $\Pr(\nexists (\det A_{\star k}^{\circ 2})^{-1} \mod p) \leq (k^2 - 1)/p$ and $\Pr(\nexists (\det A_{\star k}^{\circ 2})^{-1} \mod q) \leq (k^2 - 1)/q$. Therefore we have $\Pr(\nexists (\det A_{\star k}^{\circ 2})^{-1} \mod pq) \leq (k^2 - 1)(1/p + 1/q)$.

It remains to prove that $\det A_{\star k}^{\circ 2}$ is not identically zero as a polynomial over $\boldsymbol{A}$ 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, $\det A_{\star k}^{\circ 2}$ is identically zero if and only if it has rank less than $\binom{k+1}{2}$ for all $\boldsymbol{A}$. That is, there exist $\lambda_{ij}(\boldsymbol{A}) \in \mathbb{Z}_p$ $(0 \leq i \leq j < k)$, not all zero, so that

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

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

Clearly $\lambda_{k-1, k-1} = 0$. Otherwise, whatever $\boldsymbol{\alpha}, \boldsymbol{\beta}$, we can choose values for $\boldsymbol{a}_k$ 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 $\boldsymbol{a}_{\star i}$ $(0 \leq i < k - 1)$ contains A_{k-1} as a submatrix, which has rank $(k - 1)$ with high probability by Lemma 6. Thus $\boldsymbol{\beta} \neq \boldsymbol{0}$ and, whatever $\boldsymbol{\alpha}$, we can choose values for $\boldsymbol{a}_k$ 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 $\hat{A}_\star^{\circ 2}$ with $\binom{k}{2}$ columns $\boldsymbol{a}_{\star i} \circ \boldsymbol{a}_{\star j}$ $(0 \leq i \leq j < k - 1)$ contains $A_{\star, k-1}^{\circ 2}$ as a submatrix, and therefore has rank $\binom{k}{2}$ by induction. Hence $\alpha \neq \boldsymbol{0}$, implying $\mathcal{L} \neq 0$, a contradiction. $\quad\square$

Lemma 9. $\Pr(\det C = 0 \mod pq) \leq (2k - 1)(1/p + 1/q)$.

Proof. From Lemma 6, $\det A = 0 \mod p$ 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 $\mod pq$. 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 [38], $\Pr(\det W' = 0 \mod p) \leq k/p$ and $\Pr(\det W' = 0 \mod q) \leq k/q$. So $\det W'$ is zero or a divisor of zero $\mod pq$ with probability at most $k(1/p + 1/q)$. So $\det A \det W' = 0 \mod pq$ with probability at most $(2k - 1)(1/p + 1/q)$. So $\det C \neq 0$ with high probability. $\quad\square$

Lemma 10. $\Pr(\det C_0 = 0 \mod pq) \leq (2k - 1)(1/p + 1/q)$.

Proof. Note that $C_0 = C$ if $m_1 = m_2 = \cdots = m_k = 0$. Since Lemma 9 holds in that case, the result follows. $\quad\square$

Lemma 11. (7) *holds if and only if*

$$\sum_{t=1}^{k-1} \sigma_{jlt} \varrho_{it} = \sum_{t=1}^{k-1} \sigma_{ijt} \varrho_{lt} \pmod{q}, \ \forall i, j, l \in [1, k-1].$$

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

$$\boldsymbol{a}_i \cdot (\boldsymbol{a}_j \cdot \boldsymbol{a}_l) = \boldsymbol{a}_i \cdot \left(p \varrho_{jl} \boldsymbol{1} + \sum_{t=1}^{k-1} \sigma_{jlt} \boldsymbol{a}_t\right)$$
$$= p \varrho_{jl} \boldsymbol{a}_i + \sum_{t=1}^{k-1} \sigma_{jlt} \boldsymbol{a}_i \cdot \boldsymbol{a}_t,$$

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

Lemma 12. *Let* $\tau, \varrho_i \stackrel{\$}{\leftarrow} [0, q)$ $(i \in [1, k-1])$, *let* $\varrho_{ij} = \varrho_i \varrho_j \mod q$, *and let the* σ_{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* $i, j, \ell \in [1, k-1]$, $\boldsymbol{\gamma}^T(\boldsymbol{a}_i \cdot (\boldsymbol{a}_j \cdot \boldsymbol{a}_l)) = \tau \varrho_i \varrho_j \varrho_l \mod q$, *the symmetry of which implies that* (7) *holds.*

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

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

$\square$

B Derivation of bounds

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 [16, 31].

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

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

So, we may take

$$
\begin{aligned}
p = 2^\lambda &> M^{3d/2} \approx 2^{3d\rho/2} \\
\text{i.e. } \lambda &\approx 3d\rho/2 \\
\text{and } \eta &\approx \frac{\lambda^2}{\rho} - \lambda = \frac{3d\lambda}{2} - \lambda = \frac{3d\rho}{2}\left(\frac{3d}{2} - 1\right)
\end{aligned}
$$

For HE1N, we assume $M \approx 2^\rho$, and we have $\rho' = \rho + \lg \kappa$. Now,

$$
\begin{aligned}
\kappa &> (n+1)^d M^d \approx (nM)^d \text{ for large } n, \\
\text{i.e. } \lg \kappa &\approx d(\lg n + \rho)
\end{aligned}
$$

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

$$
\begin{aligned}
\lg \kappa &> d\lg n + d(\rho' - \lg \kappa) \\
\text{i.e. } \lg \kappa &\approx \frac{d(\lg n + \rho')}{d+1}
\end{aligned}
$$

Since κ is much larger than M, we also have

$$
\begin{aligned}
p = 2^\lambda &> (n+1)^d(M + \kappa^2)^d \approx (n\kappa^2)^d \text{ for large } n \\
\text{i.e. } \lambda &\approx d(\lg n + 2\lg \kappa), \\
\text{and } \eta &\approx \frac{\lambda^2}{\rho'} - \lambda = \frac{3d\lambda}{2} - \lambda = \frac{3d\rho'}{2}\left(\frac{3d}{2} - 1\right)
\end{aligned}
$$

Then we can calculate η as for HE1 above. Note that, in both HE1 and HE1N, λ scales linearly with d, and η scales quadratically.