Fully Homomorphic SIMD Operations

N.P. Smart[1] and F. Vercauteren[2]

[1] Dept. Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
nigel@cs.bris.ac.uk
[2] COSIC - Electrical Engineering,
Katholieke Universiteit Leuven,
Kasteelpark Arenberg 10,
B-3001 Heverlee,
Belgium.
fvercaut@esat.kuleuven.ac.be

Abstract. At PKC 2010 Smart and Vercauteren presented a variant of Gentry's fully homomorphic public key encryption scheme and mentioned that the scheme could support SIMD style operations. The slow key generation process of the Smart–Vercauteren system was then addressed in a paper by Gentry and Halevi, but their key generation method appears to exclude the SIMD style operation alluded to by Smart and Vercauteren. In this paper, we show how to select parameters to enable such SIMD operations, whilst still maintaining practicality of the key generation technique of Gentry and Halevi. As such, we obtain a somewhat homomorphic scheme supporting both SIMD operations and operations on large finite fields of characteristic two. This somewhat homomorphic scheme can be made fully homomorphic in a naive way by recrypting all data elements seperately. However, we show that the SIMD operations can be used to perform the recrypt procedure in parallel, resulting in a substantial speed-up. Finally, we demonstrate how such SIMD operations can be used to perform various tasks by studying two use cases: implementing AES homomorphically and encrypted database lookup.

1 Introduction

For many years a long standing open problem in cryptography has been the construction of a fully homomorphic encryption (FHE) scheme. The practical realisation of such a scheme would have a number of consequences, such as computation on encrypted data held on an untrusted server. In 2009 Gentry [8, 9] came up with the first construction of such a scheme based on ideal lattices. Soon after Gentry's initial paper appeared, two other variants were presented [5, 16]; the method of van Dijk et al. [5] is a true variant of Gentry's scheme and relies purely on the arithmetic of the integers; on the other hand the scheme of Smart and Vercauteren [16] is a specialisation of Gentry's scheme to a particular set of parameters.

All schemes make use of Gentry's idea of first producing a somewhat homomorphic encryption scheme and then applying a bootstrapping process to obtain a complete FHE scheme. This bootstrapping process requires a "dirty" ciphertext to be publicly reencrypted into a "cleaner" ciphertext. This requires that the somewhat homomorphic scheme can homomorphically implement its own decryption circuit, and so must be able to execute a circuit of a given depth.

Recently, Gentry and Halevi [10] presented an optimized version of the Smart–Vercauteren variant. In particular, the optimized version has an efficient key generation procedure based on the Fast Fourier Transform and a simpler decryption circuit. These two major optimizations, along with some other minor ones, allow Gentry and Halevi to actually implement a "toy" FHE scheme, including the ciphertext cleaning operation.

Smart and Vercauteren mentioned in [16] that their scheme can be adapted to support SIMD (Single-Instruction Multiple-Data) style operations on non-trivial finite fields of characteristic two, as opposed to operations on single bits, as long as the parameters are chosen appropriately. However, the parameters proposed in both [10] and [16] do not allow such SIMD operations, nor direct operation on elements of finite fields of characteristic two of degree greater than one. In particular, the efficient key generation method of [10] precludes the use of parameters which would support SIMD style operations. Using fully homomorphic SIMD operations would be an advantage in any practical system since FHE schemes usually embed relatively small plaintexts within large ciphertexts. Allowing each ciphertext to represent a number of independent plaintexts would therefore enable more efficient use of both space and computational resources.

In this paper we investigate the use of SIMD operations in FHE systems in more depth. In particular we show how by adapting the parameter settings of [10, 16] one can obtain the benefits of SIMD operations, whilst still maintaining many of the important efficiency improvements obtained by Gentry and Halevi. We thus obtain a somewhat homomorphic scheme supporting SIMD operations, and operations on large finite fields of characteristic two. We then discuss how one can use the SIMD operations to perform the recrypt procedure in parallel. In addition we explain how such SIMD operations could be utilized to perform a number of interesting higher level operations, such as performing AES encryption homomorphically and searching an encrypted database on a remote server.

The paper is structured as follows. Section 2 presents some basic facts about finite fields and algebras defined as quotients of polynomial rings. Section 3 explains how these algebras allow us to create a somewhat homomorphic encryption scheme whose message space consists of multiple parallel copies of a given finite field of characteristic two. Section 4 describes a recryption procedure for the somewhat homomorphic scheme that preserves the underlying message space structure. Section 5 contains our main contribution, namely, a recryption procedure that makes use of the SIMD operations. This new procedure significantly reduces the cost of recryption. To justify our claims, Section 6 presents implementation timings for a toy example. Finally, Section 7 gives possible applications of the SIMD structure of our FHE scheme, including bit-sliced implementations of algorithms, such as performing AES encryption using an encrypted key, and database search.

Notations We end this introduction by presenting the notations that will be used throughout this paper. Assignment to variables will be denoted by $x \leftarrow y$. If A is a set then $x \leftarrow A$ implies that x is selected from A using the uniform distribution. If A is an algorithm then $x \leftarrow A$ implies that x is obtained from running A, with the resulting probability distribution being induced by the random coins of A. For integers x, d, we denote $[x]_d$ the reduction of x modulo d into the interval $[-d/2, d/2)$. If $\mathbf{y}$ is a vector then we let $\mathbf{y}_i$ denote the i'th element of $\mathbf{y}$.

Polynomials over an indeterminate X will (usually) be denoted by uppercase roman letters, e.g. $F(X)$. We make an exception for the cyclotomic polynomials which are as usual denoted by $\Phi_m(X)$. Elements of finite fields and number fields defined by a polynomial $F(X)$, i.e. elements of $\mathbb{F}_2[X]/F(X)$ and $\mathbb{Q}[X]/F(X)$, can also be represented as polynomials in some fixed root of $F(X)$ in the algebraic closure of the base field. We shall denote such polynomials by lower case greek letters, with the fixed root (being an element of the field) also being denoted by a lower case greek letter; for instance $\gamma(\theta)$ where $F(\theta) = 0$. When the underlying root of $F(X)$ is clear we shall simply write γ.

For a polynomial $F(X) \in \mathbb{Q}[X]$ we let $\|F(X)\|_\infty$ denote the ∞-norm of the coefficient vector, i.e. the maximum coefficient in absolute value. Similarly, for an element $\gamma \in \mathbb{Q}[X]/F(X)$ we write $\|\gamma\|_\infty$ for $\|\gamma(X)\|_\infty$ where $\gamma(X)$ is the corresponding unique polynomial of degree $< \deg(F)$. If $F(X) \in \mathbb{Q}[X]$ then we let $\lceil F(X) \rfloor$ denote the polynomial in $\mathbb{Z}[X]$ obtained by rounding the coefficients of $F(X)$ to the nearest integer. Similary, for an element $\gamma \in \mathbb{Q}[X]/F(X)$ we write $\lceil \gamma \rfloor$ for $\lceil \gamma(X) \rfloor$.

2 Fields and Homomorphisms

To present the SIMD operations in full generality and to understand how they can be utilized we first set up a number of finite fields and homomorphisms between them. We let $F(X) \in \mathbb{F}_2[X]$ denote a monic polynomial of degree N that we assume to split into exactly r *distinct* irreducible factors of degree $d = N/r$

$$F(X) := \prod_{i=1}^{r} F_i(X).$$

In practice $F(X)$ will be the reduction modulo two of a specially chosen monic *irreducible* polynomial over $\mathbb{Z}$. This polynomial $F(X)$ defines a number field $\mathbb{K} = \mathbb{Q}(\theta) = \mathbb{Q}[X]/(F)$, where θ is some fixed root in the algebraic closure of $\mathbb{Q}$.

Let A denote the algebra $A := \mathbb{F}_2[X]/(F)$, then by the Chinese Remainder Theorem we have the natural isomorphisms

$$\begin{aligned} A &\cong \mathbb{F}_2[X]/(F_1) \otimes \cdots \otimes \mathbb{F}_2[X]/(F_r), \\ &\cong \mathbb{F}_{2^d} \otimes \cdots \otimes \mathbb{F}_{2^d}\ , \end{aligned}$$

i.e. A is isomorphic to r copies of the finite field $\mathbb{F}_{2^d}$. Arithmetic in A will be defined by polynomial arithmetic in the indeterminate X modulo the polynomial $F(X)$. Our goal in this section is to relate arithmetic in A explicitly with the elements in subfields of the $\mathbb{F}_{2^d}$.

We let θ_i denote a fixed root of $F_i(X)$ in the algebraic closure of $\mathbb{F}_2$. To aid notation we define $\mathbb{L}_i := \mathbb{F}_2[X]/(F_i)$ and note that all the $\mathbb{L}_i$ are isomorphic as fields, where the isomorphisms are explicitly given by

$$\Lambda_{i,j} : \begin{cases} \mathbb{L}_i \longrightarrow \mathbb{L}_j \\ \alpha(\theta_i) \longmapsto \alpha(\rho_{i,j}(\theta_j)) , \end{cases}$$

with $\rho_{i,j}(\theta_j)$ a fixed root of F_i in $\mathbb{L}_j$, i.e. we have $F_i(\rho_{i,j}(X)) \equiv 0 \pmod{F_j(X)}$.

For each divisor n of d, the finite field $\mathbb{K}_n := \mathbb{F}_{2^n}$ is contained in $\mathbb{F}_{2^d}$. We assume a fixed canonical representation for $\mathbb{K}_n$ as $\mathbb{F}_2[X]/K_n(X)$ for some irreducible polynomial $K_n(X) \in \mathbb{F}_2[X]$ of degree n, which is often fixed by the application. We let ψ denote a fixed root of $K_n(X)$ in the algebraic closure of $\mathbb{F}_2$. Since $\mathbb{K}_n$ is contained in each of $\mathbb{L}_i$ defined above, we have explicit homomorphic embeddings given by

$$\Psi_{n,i} : \begin{cases} \mathbb{K}_n \longrightarrow \mathbb{L}_i \\ \alpha(\psi) \longmapsto \alpha(\sigma_{n,i}(\theta_i)) , \end{cases}$$

with $\sigma_{n,i}(\theta_i)$ a fixed root of $K_n(X)$ in $\mathbb{L}_i$, i.e. $K_n(\sigma_{n,i}(X)) \equiv 0 \pmod{F_i(X)}$. Note that the above mapping is linear in the coefficients of $\alpha(\psi)$.

Combining the above homomorphic embedding with the Chinese Remainder Theorem, we obtain a homomorphic embedding of $l \leq r$ copies of $\mathbb{K}_n$ into the algebra A via

$$\Gamma_{n,l} : \begin{cases} \mathbb{K}_n^l \longrightarrow A \\ (\kappa_1(\psi), \ldots, \kappa_l(\psi)) \longmapsto \sum_{i=1}^{l} \kappa_i(\sigma_{n,i}(X)) \cdot H_i(X) \cdot G_i(X), \end{cases} .$$

The polynomials $H_i(X)$ and $G_i(X)$ are given by the Chinese Remainder Theorem and are defined as

$$H_i(X) \leftarrow F(X)/F_i(X) \text{ and } G_i(X) \leftarrow 1/H_i(X) \pmod{F_i(X)}.$$

We shall denote component wise addition and multiplication of elements in $\mathbb{K}_n^l$ by $\mathbf{k}_1 + \mathbf{k}_2$ and $\mathbf{k}_1 \times \mathbf{k}_2$. As such we have constructed two equivalent methods of computing with elements in $\mathbb{K}_n^l$: the first method simply computes component wise on vectors of l elements in $\mathbb{K}_n$, whereas the second method first maps all inputs to the algebra A using $\Gamma_{n,l}$, performs computations in A and finally maps back to $\mathbb{K}_n^l$ via $\Gamma_{n,l}^{-1}$. Note that by construction $\mathbb{K}_n^l$ and $\Gamma_{n,l}(\mathbb{K}_n^l)$ are isomorphic, so that $\Gamma_{n,l}^{-1}$ is always well defined on the result of the computation.

The goal of this paper is to produce a fully homomorphic encryption scheme that allows us to work via SIMD operations on l copies of $\mathbb{K}_n$ at a time, for all n dividing d, by computing in the algebra A. In particular, this enables us to support SIMD operations both in $\mathbb{F}_2$ and $\mathbb{F}_{2^d}$. To make things concrete the reader should consider the example of $F(X)$ being the 3485-th cyclotomic polynomial. In this situation the polynomial $F(X)$ has degree $N = \varphi(3485) = 2560$, and modulo two it factors into 64 polynomials each of degree 40. This polynomial therefore allows us to compute in parallel with up to 64 elements of any subfield of $\mathbb{F}_{2^{40}}$. For instance, by selecting $n = 1$ and $l = 64$ we perform 64 operations in $\mathbb{F}_2$ in parallel; selecting $n = 40$ and $l = 1$ we perform operations in a single copy of the finite field $\mathbb{F}_{2^{40}}$; whereas selecting $n = 8$ and $l = 16$ we perform SIMD operations on what is essentially the AES state matrix, namely 16 elements of $\mathbb{F}_{2^8}$.

3 Somewhat Homomorphic Scheme Supporting SIMD Operations in $\mathbb{K}_n$

In this section, we recall the Smart–Vercauteren variant of Gentry's somewhat homomorphic scheme and show that it can support SIMD operations in r copies of the finite field $\mathbb{K}_n$ by modifying key generation. Note that the recent FHE schemes based on ring-LWE [2] also support such style operations, and may be preferable in practice due to their improved key generation procedures, we leave it to the reader to extend our work to these new schemes. However, whilst our SIMD style operations extend to the ring-LWE based somewhat homomorphic schemes, our parallel recryption step does not carry over. We will return to this point later on.

3.1 Smart-Vercauteren somewhat homomorphic scheme

Let $F \in \mathbb{Z}[X]$ be a monic irreducible polynomial of degree N and let $\mathbb{K} = \mathbb{Q}(\theta) = \mathbb{Q}[X]/(F)$ denote the number field defined by F. Gentry's original scheme uses two co-prime ideals I and J in the number ring $\mathbb{Z}[\theta]$. The ideal I is chosen to have small norm $\mathcal{N}(I) = \sharp(\mathbb{Z}[\theta]/I)$ and determines the plaintext space, namely $\mathbb{Z}[\theta]/I$. For this reason, $I = (2)$ is chosen in practice. Note that in the case of a general F the quotient ring $\mathbb{Z}[\theta]/(2)$ is an algebra of a somewhat more general type than discussed in Section 2. We shall choose F later on such that one obtains precisely the type of algebra considered in Section 2. The ideal J determines the private/public key pair: the private key consists of a "good" representation of J, whereas the public key consists of a "bad" representation of J.

To clarify the notions of "good" and "bad", we first describe the Smart–Vercauteren instantiation. The ideal J is chosen to be principal, i.e. generated by one element $\gamma \in \mathbb{Z}[\theta]$, and has the following additional property: let $d = \mathcal{N}(J) = \sharp(\mathbb{Z}[\theta]/J) = |N_{\mathbb{K}/\mathbb{Q}}(\gamma)|$, where $N_{\mathbb{K}/\mathbb{Q}}(\cdot)$ denotes the number field norm of $\mathbb{K}$ to $\mathbb{Q}$, then there must exist a unique $\alpha \in \mathbb{Z}_d$ such that

$$J = (\gamma) = (d, \theta - \alpha) .$$

The "good" representation of J (i.e. the private key) corresponds to the small generator γ, whereas the "bad" representation (i.e. public key) is $(d, \theta - \alpha)$. The additional property of J is equivalent with the requirement that the Hermite Normal Form representation of J has the following specific form

$$\begin{pmatrix} d & 0 & 0 \dots 0 \\ -\alpha & 1 & 0 \quad\; 0 \\ -\alpha^2 & 0 & 1 \quad\; 0 \\ \vdots & & \ddots \\ -\alpha^{N-1} & 0 & 0 \quad\; 1 \end{pmatrix},$$

where the entries below d in the first column are taken modulo d. Another characterisation of this property is that the ideal J simply contains an element of the form $\theta - \alpha$. This is clearly necessary since J can be generated by $(d, \theta - \alpha)$, but it is also sufficient. Indeed, since $\gamma \in J$, this implies that $d \in J$, so $(d, \theta - \alpha) \subset J$ and since both ideals have the same norm, we must have $J = (d, \theta - \alpha)$. As such, there exists an element $\nu \in \mathbb{Z}[\theta]$ with $\nu \cdot \gamma = \theta - \alpha$. To derive an easy verifiable condition on γ, we define the algebraic number $\zeta \in \mathbb{Z}[\theta]$ such that

$$\zeta \cdot \gamma = d . \tag{1}$$

Multiplying $\nu \cdot \gamma = \theta - \alpha$ on both sides with ζ gives the condition $d \cdot \nu = \theta \cdot \zeta - \alpha \cdot \zeta$. Write $\zeta = \sum_{i=0}^{N-1} \zeta_i \cdot \theta^i$ and $F(X) = \sum_{i=0}^{N} F_i \cdot X^i$, then computing the product $\theta \cdot \zeta$ explicitly and reducing modulo d finally leads to:

$$\alpha \cdot \zeta_i = \zeta_{i-1} - \zeta_{N-1} F_i \bmod d , \tag{2}$$

for all $i = 0, \dots, N-1$ where $\zeta_{-1} = 0$.

Note that the two element representation $(d, \theta - \alpha)$ defines an easily computable homomorphism

$$H : \mathbb{Z}[\theta] \to \mathbb{Z}_d : \eta = \sum_{i=0}^{N-1} \eta_i \cdot \theta^i \mapsto H(\eta) = \sum_{i=0}^{N-1} \eta_i \cdot \alpha^i \bmod d . \tag{3}$$

The homomorphism H also makes it very easy to test if an element $\eta \in \mathbb{Z}[\theta]$ is contained in the ideal J, namely $\eta \in J$ if and only if $H(\eta) = 0$. Furthermore, given the "good" representation γ, it is possible to invert H on a small subset of $\mathbb{Z}[\theta]$ as shown by the following lemma.

Lemma 1. *Let $J = (\gamma) = (d, \theta - \alpha)$ and $\zeta \cdot \gamma = d$ and let H be defined as in* (3). *Let $\eta \in \mathbb{Z}[\theta]$ with $\|\eta\|_\infty < U$, then we have*

$$\eta = H(\eta) - \left\lceil \frac{H(\eta) \cdot \zeta}{d} \right\rfloor \cdot \gamma \qquad \text{for} \qquad U = \frac{d}{2 \cdot \delta_\infty \cdot \|\zeta\|_\infty} ,$$

where $\delta_\infty = \sup \left\{ \frac{\|\mu \cdot \nu\|_\infty}{\|\mu\|_\infty \cdot \|\nu\|_\infty} : \mu, \nu \in \mathbb{Z}[\theta] \right\}$. Furthermore, for $\|\eta\|_\infty < U$ we have

$$[H(\eta) \cdot \zeta]_d = [\eta \cdot \zeta]_d = \eta \cdot \zeta . \tag{4}$$

Proof. It is easy to see that $H(\eta)-\eta$ is contained in the principal ideal generated by γ. As such, there exists a $\beta \in \mathbb{Z}[\theta]$ such that $H(\eta) - \eta = \beta \cdot \gamma$. Using $\zeta = d/\gamma$, we can write

$$\beta = \frac{H(\eta) \cdot \zeta}{d} - \frac{\eta \cdot \zeta}{d} \,. \tag{5}$$

Since β has integer coefficients, we can recover it by rounding the coefficients of the first term if the coefficients of the second term are strictly bounded by $1/2$. This shows that η can be recovered from $H(\eta)$ for $\|\eta\|_\infty < d/(2 \cdot \delta_\infty \cdot \|\zeta\|_\infty)$. Furthermore, equation (5) shows that $[H(\eta) \cdot \zeta]_d = [\eta \cdot \zeta]_d$ and since $\|\eta\|_\infty < U$, we have $[\eta \cdot \zeta]_d = \eta \cdot \zeta$.

Corollary 1. *Using the notation of Lemma 1, assume that $\|\eta\|_\infty < U/L$, then for $i = 0, \ldots, N-1$ we have*

$$-\frac{1}{2L} < \frac{H(\eta) \cdot \zeta_i}{d} - \left\lceil \frac{H(\eta) \cdot \zeta_i}{d} \right\rfloor < \frac{1}{2L} \,,$$

i.e. $H(\eta) \cdot \zeta_i/d$ is within distance $1/2L$ of an integer.

Proof. Follows directly from equation (5) and the assumption on η.

The above lemma shows that we can recover an element η from its image under H, when its norm is not too large. As such we obtain a trapdoor one way function that can be used as the basis for encryption. Using these preliminaries we are now ready to define key generation, encryption and decryption.

KEY GENERATION: Input parameters: N, t
Generate a monic irreducible polynomial $F \in \mathbb{Z}[X]$ of degree N with small coefficients, defining the number field $\mathbb{K} = \mathbb{Q}(\theta) = \mathbb{Q}[X]/(F)$. Choose an element $\gamma \in \mathbb{Z}[\theta]$ with $\gamma = 1 \bmod 2$ such that the coefficients of γ are smaller in absolute value than 2^t (at least one coefficient should be a t-bit integer). Compute the norm $d = |N_{\mathbb{K}/\mathbb{Q}}(\gamma)|$ as well as the element $\zeta \in \mathbb{Z}[\theta]$ with $\zeta \cdot \gamma = d$. If d is even, choose a new γ. If d is odd, compute $\alpha = -\zeta_{N-1} \cdot F_0/\zeta_0$ and verify whether (2) holds for all $i = 1, \ldots, N-1$. If not, generate a new γ. Otherwise, the public key is the pair $\mathsf{pk} := (d, \alpha)$ whereas the private key is the element $\mathsf{sk} := \zeta$.

In practice, N will be of the order a few thousand and t a few hundred. The size of d can be approximated roughly by $N^N \cdot 2^{Nt}$; this therefore results in a d of several million bits.

ENCRYPTION: Input parameters: μ, $\mathsf{pk} := (d, \alpha)$, message $M \in A := \mathbb{F}_2[X]/(F(X))$
The plaintext space consists of (a subalgebra of) the algebra $A := \mathbb{F}_2[X]/(F(X))$. Represent the message M as a polynomial $M(X) \in \mathbb{Z}[X]$ with coefficients in $\{0, 1\}$. Generate a "noise" polynomial $R(X) \in \mathbb{Z}[X]$ of degree $< N$ with $\|R(X)\|_\infty \le \mu$ and compute the ciphertext as

$$c \leftarrow [M(\alpha) + 2 \cdot R(\alpha)]_d \,.$$

Note that the ciphertext is an element in $\mathbb{Z}_d$ and that encryption simply corresponds to applying the homomorphism H to the algebraic integer $C(\theta) := M(\theta) + 2 \cdot R(\theta)$. Furthermore, it should be clear that if we can recover $C(\theta)$, then we can decrypt simply by computing $C(X) \bmod 2$. The encryption function is denoted as $c \leftarrow \mathsf{Encrypt}(M(X), \mathsf{pk})$. If $M(X) \in A$ then we say $M\big|_\alpha = M(\alpha) \pmod d$ is a "trivial" encryption of $M(X)$, i.e. it is an encryption with no randomness.

DECRYPTION: Input parameters: ciphertext $c \in \mathbb{Z}_d$, $\mathsf{sk} := \zeta$
Given the ciphertext $c \in \mathbb{Z}_d$, compute the element $C(\theta)$ as

$$C(\theta) = c - \left\lceil \frac{c \cdot \zeta}{d} \right\rfloor \,,$$

and then set $M(X) = C(X) \bmod 2$. Note that here we used the fact that $\gamma \equiv 1 \bmod 2$. We can obtain a simpler decryption procedure using the last statement in Lemma 1. Indeed, if c is a decryptable ciphertext, we know that $\|C(\theta)\|_\infty < U$ and thus that

$$[c \cdot \zeta]_d = C(\theta) \cdot \zeta \,.$$

Since $\gamma \equiv 1 \bmod 2$ and d is odd with $d = \gamma \cdot \zeta$, we see that also $\zeta \equiv 1 \bmod 2$. Furthermore, $C(\theta) = M(\theta) + 2R(\theta)$, so we obtain

$$[c \cdot \zeta]_d \bmod 2 = M(\theta) \bmod 2 = M(X) \,.$$

This shows that for $\zeta = \sum_{i=0}^{N-1} \zeta_i \theta^i$ we can recover the coefficients of $M(X) = m_0 + m_1 \cdot X + \cdots + m_{N-1} \cdot X^{N-1}$ one by one, by computing

$$m_i = [c \cdot \zeta_i]_d \pmod 2 .$$

We write $M(X) \leftarrow \mathsf{Decrypt}(c, \mathsf{sk})$. Note that to save space for key storage, it suffices to store ζ_0, since the other ζ_i follow from equation (2). In particular, we obtain the closed expression $\zeta_i = w_i \cdot \zeta_0$ with

$$w_i = -\frac{1}{F_0} \left(\sum_{j=i+1}^{N} F_j \cdot \alpha^{j-i} \right) \pmod d . \tag{6}$$

Since the w_i can be publicly computed, we can decrypt $m_i = [c \cdot w_i \cdot \zeta_0]_d \pmod 2$. We pause to note that it is this linear relationship between the distinct decryption keys ζ_i which enables the parallel recryption procedure we describe later. For ring-LWE based somewhat homomorphic schemes supporting SIMD operations, where such a simple linear relation does not hold, it seems much harder to produce a parallel recryption procedure using the squashing paradigm of Gentry.

HOMOMORPHIC OPERATIONS: It is easy to see that the scheme is somewhat homomorphic, where the operations being performed are addition and multiplication of ciphertexts modulo d. Indeed, let $c_i = H(C_i(\theta)) = H(M_i(\theta) + 2R_1(\theta))$ for $i = 1, 2$, then we have that

$$\begin{aligned} c_1 + c_2 &= H(M_1(\theta) + M_2(\theta) + 2(R_1(\theta) + R_2(\theta))) \\ c_1 \cdot c_2 &= H(M_1(\theta) \cdot M_2(\theta) + 2(M_1(\theta)R_2(\theta) + M_2(\theta)R_1(\theta) + 2R_1(\theta)R_2(\theta))) . \end{aligned}$$

This shows that operations on the ciphertext space induce corresponding operations on the plaintext space, i.e. the algebra A. Thus it is clear that the somewhat homomorphic scheme supports SIMD operations and operations on elements in possibly large degree (i.e. degree n) finite fields. To make a distinction when we are performing homomorphic operations we will use the notation $\oplus$ and $\odot$ to denote the homomorphic addition and multiplication of ciphertexts.

3.2 Efficient key generation and SIMD operations

Whilst the FHE scheme works for any polynomial F with small coefficients, the common case, as in [10] and [16], is to use the polynomial $F(X) := X^{2^n} + 1$. As pointed out by Gentry and Halevi [10] this leads to a major improvement in the key generation procedure over that proposed by Smart and Vercauteren [16]. If we let η_i denote the roots of the polynomial F over the complex numbers, or over a sufficiently large finite field, then we can compute ζ and d as follows:

- Compute $\omega_i \leftarrow \gamma(\eta_i) \in \mathbb{C}$ for all i.
- Compute $d \leftarrow \prod \omega_i$.
- Compute $\omega_i^* \leftarrow 1/\omega_i$.
- Interpolate the polynomial ζ/d from the data values ω_i^*.

The key observation is that since $F(X)$ is of the form $X^{2^n} + 1$, the η_i are 2^{n+1}-th roots of unity and so to perform the polynomial evaluation and interpolation above we can apply the Fast Fourier Transform (FFT). Indeed, Gentry and Halevi present an even more optimized scheme to compute d and ζ which requires only polynomial arithmetic, but this makes significant use of the fact that the trace of 2-power roots of unity is always zero.

The problem with selecting $F(X) = X^{2^n} + 1$ is that it has only one irreducible factor modulo two. In particular if we select $F(X) = X^{2^n} + 1$ then the underlying plaintext algebra is given by

$$A := \mathbb{F}_2[X]/(F) \cong \mathbb{F}_2[X]/(X-1)^{2^n} .$$

In other words, F does not split into a set of *distinct* irreducible factors modulo two as we required to enable SIMD operations.

We now present a possible replacement for $F(X)$. The key observation is that we need an $F(X)$ which enables fast key generation via FFT like algorithms, which has small coefficients, and which splits into distinct irreducible factors modulo two of the same degree. In addition we need a relatively large supply of such polynomials to cope with increasing security levels (i.e. N), different numbers of parallel operations (i.e. l) and different degree two finite fields in which operations occur (i.e. n). In particular need to pick an $F(X)$ which generates a Galois extension of degree n. In addition we need to select a polynomial $F(X)$ such that 2 is neither ramified, nor an index divisor, in the associated number field generated by a root of $F(X)$. These conditions ensure that the algebra mod two splits into distinct finite fields of the same degree.

One is then led to consider other cyclotomic polynomials as follows. We select an odd integer m and recall that the m-th cyclotomic polynomial is defined by

$$\Phi_m(X) := \prod_{\eta}(X - \eta)$$

where η ranges over all m-th primitive roots of unity. We have $\deg(\Phi_m(X)) = \phi(m)$, and that $\Phi_m(X)$ is an irreducible polynomial with integer coefficients. In the practical range for m, the coefficients of Φ_m are very small, e.g. for all $m \leq 40000$ the coefficients are bounded by 59 and are in most cases much smaller than this upper bound.

The field $\mathbb{Q}(\theta)$ is a Galois extension and hence each prime ideal splits in $\mathbb{Q}(\theta)$ into a product of prime ideals of the same degree and ramification index. If m is odd then the prime two does not ramify in the field $\mathbb{Q}(\theta)$, nor is it an index divisor. In particular, by Dedekind's criterion, this means that the polynomial $\Phi_m(X)$, of degree $N = \phi(m)$, factors modulo two into a product of $r = N/d$ distinct irreducible polynomials of degree equal to the unique degree d of the prime ideals lying above the ideal (2). This degree d is the smallest integer such that $2^d \equiv 1 \pmod{m}$.

Hence, by selecting $F(X) := \Phi_m(X)$ in our construction of the algebra A over $\mathbb{F}_2$, we find that A is isomorphic to a product of r finite fields of degree $d = N/r$. The only issue is whether one can perform the key generation efficiently. To do this we use Fourier Transforms with respect to the m-th roots of unity. In particular given the polynomial γ in the key generation procedure we compute the evaluation at the m-th roots of unity via a Fourier Transform, and produce the norm d by selecting the N required values to multiply together (consisting of the evaluations of the primitive roots of unity). One can then compute $1/\gamma$ by inverting the Fourier coefficients and then interpolating via the inverse Fourier Transform.

In other words the same optimization as mentioned earlier can be applied: Instead of taking the standard Cooley-Tukey [6] FFT method for powers of two, we apply the Good-Thomas method [11, 18] for when m is a product of two coprime integers, or Cooley-Tukey when m is a prime power. Either method reduces the problem to computing FFTs for prime power values of m, for which we can use the Rader FFT algorithm [15]. This in itself reduces the problem to computing a convolution of two sequences, which is then performed by extension of the sequences to length a power of two followed by the application of the Cooley-Tukey algorithm to the extended sequence. Overall the FFT then takes $O(m \cdot \log m)$ operations on elements of size $O(\log_2 d)$ bits. In practice $m \approx 2 \cdot N$ and so this gives the same complexity for key generation as using $F(X) = X^{2^n} + 1$, however the implied constants are slightly greater. This means we can achieve almost the same complexity for key generation as in the 2-power root of unity case.

4 Fully Homomorphic Scheme and Naive Recryption Method

To turn the somewhat homomorphic scheme of the previous section into a fully homomorphic scheme, we follow Gentry's bootstrapping approach, i.e. we squash the decryption circuit so much that it can be evaluated by the somewhat homomorphic scheme. In particular, we use the optimized procedure described by Gentry and Halevi in [10].

4.1 Recryption Method

Recall that each message bit m_i can be recovered as $m_i = [c \cdot w_i \cdot \zeta_0]_d \pmod{2}$ with the w_i being publicly computable constants defined in (6). Since $[c \cdot w_i]_d$ can be computed without knowledge of ζ_0 it suffices to show how $[c \cdot \zeta_0]_d \pmod{2}$ can be computed with a low complexity circuit.

The idea is to write the private key ζ_0 as the solution to a sparse-subset-sum problem. In particular, we will define s sets of S elements as follows (a discussion on the sizes of s and S will be given later): choose s elements $x_i \in [0, \ldots, d)$, a random integer $R \in [1, \ldots, d)$ and define the i-th set $\mathcal{B}_i = \{x_i \cdot R^j \pmod{d} \mid j \in [0, \ldots, S)\}$ such that the private key ζ_0 can be written as the sum

$$\zeta_0 = \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot x_i \cdot R^j \pmod{d},$$

where for each i only one $b_{i,j} = 1$ and all other $b_{i,j}$ are zero. The index j for which $b_{i,j} = 1$ will be denoted by e_i and so we can write $\zeta_0 = \sum_{i=1}^{s} x_i \cdot R^{e_i} \pmod{d}$. The result is that we have written ζ_0 as the sum of s elements, where one element is taken from each $\mathcal{B}_i$. To enable recryption or ciphertext cleaning, we will augment the public key with additional information: compute the ciphertexts $c_{i,j} \leftarrow \mathsf{Encrypt}(b_{i,j}, \mathsf{pk})$ for $1 \le i \le s,\ 0 \le j < S$, then the public key now consists of the data

$$\left(d, \alpha, s, S, R, \{x_i, \{c_{i,j}\}_{j=0}^{S-1}\}_{i=1}^{s}\right).$$

Denote $y_{i,j} = c \cdot x_i \cdot R^j \pmod{d}$ for $i = 1, \ldots, s$ and $j = 0, \ldots, S-1$ such that $0 \le y_{i,j} < d$, then the decryption function $[c \cdot \zeta_0]_d \pmod{2}$ can be rewritten as

$$\begin{aligned}
[c \cdot \zeta_0]_d \pmod{2} &= \left[\sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot y_{i,j}\right]_d \pmod{2} \\
&= \left(\sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot y_{i,j}\right) - d \cdot \left\lfloor \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot \frac{y_{i,j}}{d} \right\rceil \pmod{2} \\
&= \bigoplus_{i=1}^{s} \bigoplus_{j=0}^{S-1} b_{i,j} \cdot y_{i,j} \pmod{2} \oplus \left\lfloor \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot \frac{y_{i,j}}{d} \right\rceil \pmod{2}.
\end{aligned}$$

Note that the latter double sum $\mathcal{T} = \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot \frac{y_{i,j}}{d}$ is equal to $c \cdot \zeta_0 / d$ and if we assume that c is the image of $C(\theta)$ under H, where $\|C(\theta)\|_\infty < U/(s+1)$, then we know by Corollary 1 that $\mathcal{T}$ is within distance $1/2(s+1)$ of an integer. If we now replace each $\frac{y_{i,j}}{d}$ with an approximation $z_{i,j}$ up to p bits after the binary point, i.e. $|z_{i,j} - y_{i,j}/d| < 2^{-(p+1)}$, then since there are only s non-zero terms, we have that $|\mathcal{T} - \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot z_{i,j}| < s \cdot 2^{-(p+1)}$. Rounding the double sum over the $z_{i,j}$ will thus give the same result as rounding $\mathcal{T}$ as long as

$$\frac{1}{2(s+1)} + s \cdot 2^{-(p+1)} < 1/2\,,$$

which implies that $p \ge \lceil \log_2(s+1) \rceil$. Furthermore, in the inner sum we are adding S numbers of which only one is non-zero. As such, we can compute the k-th bit of this sum by simply XOR-ing the k-th bits of the $b_{i,j} \cdot z_{i,j}$ for $j = 1, \ldots, S$. We are then left with an addition of s numbers, each which consists of p bits after the binary point.

We are now ready to formulate the recrypt algorithm by mapping these equations into the encrypted domain. To this end, we require two helper functions. The first function $\mathbf{b} \leftarrow \mathsf{compute\_bits}(y)$ takes as input an integer $0 \le y < d$ and outputs the vector of bits $\mathbf{b} = (b_0, b_1, \ldots, b_p)$ such that

$$\left|\frac{y}{d} - (b_0 + \frac{b_1}{2} + \frac{b_2}{2^2} + \cdots + \frac{b_p}{2^p})\right| < \frac{1}{2^{p+1}}.$$

This is easily computed by determining $u \leftarrow \lfloor (2^p \cdot y)/d \rceil$, and then reading the bits from the (small) integer u.

The second function $\mathsf{school\_book\_add}(A)$ takes as input an $s \times (p+1)$ array A of ciphertexts, where each row contains the encryptions of the $(p+1)$ bits of an integer. The result of the function is a $(p+1)$ vector containing the encryptions of the $(p+1)$ bits of the sum of these s integers modulo 2^{p+1}. The school book method is discussed in more detail in [10] where it is shown that it requires

$$T_{\mathsf{school\_book\_add}} := \left(s \cdot 2^{p-1} + \sum_{k=1}^{p-1} (s+k) \cdot 2^{p-k}\right) \cdot T_{\mathsf{mod},d}$$

where $T_{\mathsf{mod},d}$ denotes the cost of one multiplication modulo d.

In Algorithm 1 we present the algorithm for recrypting the first bit of the message underlying a ciphertext c, i.e. the algorithm computes $[c \cdot \zeta_0]_d \pmod{2}$ in the encrypted domain using the augmented public key. This is essentially the recryption algorithm used by Gentry and Halevi, where the message space is one bit only. To obtain the recyption of the i-th coefficient we simly input $[c \cdot w_i]_d$ instead of c, since decrypting the i-th bit is given by $[c \cdot w_i \cdot \zeta_0]_d \pmod{2}$.

Algorithm 1: $\mathsf{BitRecrypt}(c, \mathsf{pk})$: Recrypting the First Bit of the Plaintext Associated With Ciphertext c

$A \leftarrow 0$, where $A \in M_{s\times(p+1)}(\mathbb{Z}_d)$.
sum $\leftarrow 0$.
for i **from** 1 **upto** s **do**
  $y \leftarrow c \cdot x_i \pmod d$.
  for j **from** 0 **upto** $S-1$ **do**
    if y is odd **then**
      sum $\leftarrow$ sum $\oplus\, c_{i,j}$.
    $\mathbf{b} \leftarrow \mathsf{compute\_bits}(y)$.
    for u **from** 0 **upto** p **do**
      $A_{i,u} \leftarrow A_{i,u} \oplus (\mathbf{b}_u \cdot c_{i,j})$.
    $y \leftarrow y \cdot R \pmod d$.
$\mathbf{a} \leftarrow \mathsf{school\_book\_add}(A)$.
$\overline{c} \leftarrow$ sum $\oplus\, \mathbf{a}_0$.
return $(\overline{c})$.

We denote the cost of executing this algorithm for a one bit ciphertext as T_{bits}. Ignoring the modular additions, we see that $T_{\mathsf{bits}} = \Big((S+1)\cdot s \cdot + s \cdot 2^{p-1} + \sum_{k=1}^{p-1}(s+k)\cdot 2^{p-k}\Big)\cdot T_{\mathsf{mod},d}$.

To recrypt a whole ciphertext c, we first form ciphertexts $\overline{c}_i = \mathsf{BitRecrypt}([c \cdot w_i]_d, \mathsf{pk})$ for $i = 0, \ldots, N-1$, which are recryptions of the coefficients of the underlying polynomial $M(X)$ by submitting $[c \cdot w_i]_d$ to Algorithm 1. Then given $\overline{c}_i$ we form the ciphertext

$$\overline{c} \leftarrow \sum_{i=0}^{N-1} \overline{c}_i \odot \alpha^i$$

which will be a recryption of the original ciphertext. Note, to control the noise this last sum is computed naively, and not via Horner's rule, i.e. we multiply each coefficient ciphertext $\overline{c}_i$ by $\alpha^i \pmod d$ and then sum. The resulting algorithm is summarized in Algorithm 2. Assuming the $\alpha^i \pmod d$ and w_i are precomputed, the total cost of recrypting a ciphertext corresponding to an arbitrary element in A (using our naive method) is essentially $N \cdot T_{\mathsf{bits}} + 2 \cdot N \cdot T_{\mathsf{mod},d}$. If SIMD style operations, and operations on larger datatypes, are to be supported we therefore need a more efficient method to perform recryption.

Algorithm 2: Recrypting Ciphertext c version 1

$\overline{c} \leftarrow 0$.
for i **from** 0 **upto** $N-1$ **do**
  $\overline{c}_i \leftarrow \mathsf{BitRecrypt}([c \cdot w_i]_d, \mathsf{pk})$.
  $\overline{c} \leftarrow \overline{c} \oplus \overline{c}_i \odot \alpha^i$.
return $(\overline{c})$.

4.2 Security Analysis and Parameters

The analysis of Gentry of the above scheme and bootstrapping operation applies in our situation. The security of the underlying somewhat homomorphic scheme is based on the hardness of a variant of the bounded distance decoding (BDDP) problem; whereas the security of the bootstrapping procedure is based on the sparse subset sum problem (SSSP). Indeed the minor modifications we make in future sections to the public key result in exactly the same security reductions. Thus an adversary against the scheme can either be turned into an algorithm to solve a decision variant of the BDDP, or a SSSP.

When selecting key sizes for cryptographic schemes, in practice one almost always selects key sizes based on the *best known attacks* and not on the hard problems to which a security problem reduces. We have various parameters we need to select s, S, N, t and μ. The sizes of N, t and μ determine whether one can break the scheme by distinguishing ciphertexts, or (more seriously) by message or key recovery. Parameter selection is here based on the hardness of solving explicit closest vector problems (CVPs), in lattices of dimension N, involving basis matrices with coefficients bounded by d (a function of t and N), and for close vectors whose distance to the lattice is related to the size of μ. An algorithm to solve the CVP/BDDP can be directly used to recover plaintexts as explained in [16]. The larger the ratio of t to μ the easier it is to recover plaintexts, but the ratio of t to μ also determines how complicated a circuit the basic somewhat homomorphic scheme can evaluate. Indeed the smaller the ratio of t to μ the less expressive our somewhat homomorphic scheme is. In selecting N, t and μ one needs to make a careful analysis of the current state of the art in lattice basis reduction; a topic which is beyond the scope of this paper.

On the other hand, it is not the case that an algorithm to solve the sparse subset sum problem can be used to break the scheme. The security proof in [9] uses the FHE adversary to solve the following SSSP

$$\zeta_0 = \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot (x_i \cdot R^j) \pmod{d}.$$

The simulator (solving SSSP) is given ζ_0 and the weights $x_i \cdot R^j \pmod{d}$, and uses random ciphertexts $c_{i,j}$ to represent the encryption of the $b_{i,j}$. Since the proof has already shown that ciphertexts of specific values are indistinguishable from encryptions of random values, the adversary does not know it is in a simulation. The proof in [9] shows how the simulator can then solve the SSSP. Whilst this easily establishes the fact that the recrypt procedure does not reduce the security of the scheme, assuming of course the scheme is KDM secure and the SSSP is hard, it actually tells us very little in practice. In particular it says: "If the adversary knows the secret key, then recovering another representation of the secret key is equivalent to solving the SSSP".

Thus the parameters s and S determine (in practice) a hidden sparse subset sum problem rather than a standard SSSP. Namely, the adversary needs to solve the above subset sum problem where he is not given access to the value ζ_0. Taking the pragmatic view of parameter selection based on the best known attack, it is clear that neither the lattice attacks on the SSSP nor the time-memory trade off methods to solve the SSSP apply in the hidden case. This has important direct implications for parameter size selection. If a time-memory trade off is possible then we need to select S and s such that $S^{\lfloor s/2 \rfloor} > 2^\lambda$, where we do not believe the adversary can perform 2^λ operations.

A more pragmatic view of parameter selection would imply that, since the time-memory trade off against the hidden SSSP appears impossible, that we select $S^s > 2^\lambda$. This has a number of direct consequences: Firstly we can select S to be much smaller than Gentry–Halevi do, secondly this means we do not need to complicate the recryption procedure with the index encoding method they use to save space, since S is now small enough to not require it. Thirdly this halves the degree of the resulting recryption circuit which makes the scheme more efficient, and fourthly it saves on the computational cost of recryption, since we need to do less work.

In summary: in practice one should select N, t and μ according to best practice from lattice basis reduction. For real systems this means that parameters need to be chosen that are significantly larger than the toy examples presented in Gentry–Halevi. However, when selecting s and S one can be less conservative than Gentry–Halevi.

In Section 5 we detail a parallel recryption procedure which has the same multiplicative depth as the one above; but which requires more addition operations, where the number of extra additions depends on the level of SIMD operations required. Thus the value of t may need to be larger than that required in non SIMD based schemes. Asymptotically the constant increase will make no difference, but for "practical" parameters one may have a noticeable difference. Thus in Section 5 we present experimental results for "toy" security levels. This is done purely to show that our algorithms make a difference even for choices of N, μ and t corresponding to low security levels.

5 Parallel Recryption

Whilst Algorithm 1 will recrypt a ciphertext that encodes an element of the algebra A, it can be made significantly more efficient. Firstly, the procedure recrypts a general element in A, yet in practice we will only have that c contains $l \cdot n \leq N$ encrypted bits. Secondly, since the recrypt procedure is a binary circuit we can run it on the r embedded copies of $\mathbb{F}_2$, i.e. we can use the SIMD style operations to recrypt r bits in parallel.

The first optimization is easy to obtain: recall that $\Gamma_{n,l}$ maps a vector of l binary polynomials $(\kappa_1(\psi), \ldots, \kappa_l(\psi))$ each of of degree less than n, into a polynomial $a(X)$ of degree less than N. The map $\Gamma_{n,l}$ thus defines an isomorphism between $\mathbb{K}_n^l$ and $\Gamma_{n,l}(\mathbb{K}_n^l)$ so $\Gamma_{n,l}^{-1}$ is well defined on the result of the computation. We can represent $\Gamma_{n,l}^{-1}$ explicitly by an $(n \cdot l) \times N$ binary matrix B over $\mathbb{F}_2$ which is defined as follows:

$$\mathsf{coeff}(\kappa_i, j) = \sum_{k=0}^{N-1} B_{j+i\cdot n+1,k+1} \cdot \mathsf{coeff}(a(X), k).$$

Using B we can therefore first obtain encryptions of all the coefficients of the κ_i, recrypt these using Algorithm 1 and then reconstruct the recrypted ciphertext using $\Gamma_{n,l}$. In particular, denote with $\overline{c}_{i_1,i_2}$ a recryption of the i_1th coefficient of the i_2th component in $\mathbb{K}_n^l$, then we can obtain a full recryption of an element in $\mathbb{K}_n^l$ by computing

$$\overline{c} \leftarrow \sum_{i_1=0}^{n-1} \sum_{i_2=1}^{l} \overline{c}_{i_1,i_2} \odot \left(\left(\Gamma_{n,l}(0,\ldots,0,\psi^{i_1},0,\ldots,0) \right) \big|_\alpha \right) ,$$

where $(0,\ldots,0,\psi^{i_1},0,\ldots,0) \in \mathbb{K}_n^l$ is the element whose i_2th component is equal to ψ^{i_1}, and $M(X)\big|_\alpha$ is the trivial encryption of the element $M(X)$ in the algebra A.

Recall that given a ciphertext c, the value $[c \cdot w_i]_d$ is an encryption of the ith coefficient of $a(X)$. Since the scheme is homomorphic and using the matrix B we conclude that

$$c_{i_1,i_2} = \left[\sum_{k=0}^{N-1} B_{i_1+i_2\cdot n+1,k+1} [c \cdot w_k]_d \right]_d = \left[c \cdot \left(\sum_{k=0}^{N-1} B_{i_1+i_2\cdot n+1,k+1} \cdot w_k \right) \right]_d$$

is a valid encryption of $\mathsf{coeff}(\kappa_{i_2}, i_1)$. Note that these quantities are obtained as the sum of maximum N ciphertexts, which implies that the original c has to be an encryption of $C(\theta)$ with $\|C(\theta)\|_\infty < U/((s+1)\cdot N)$ for Algorithm 1 to recrypt correctly. The second algorithm thus first computes the $n \cdot l$ constants (the w_i are no longer required)

$$v_{i_1,i_2} = \sum_{k=0}^{N-1} B_{i_1+i_2\cdot n+1,k+1} \cdot w_k \pmod{d} ,$$

and then computes the recryptions $\overline{c}_{i_1,i_2} = \mathsf{BitRecrypt}([c \cdot v_{i_1,i2}]_d, \mathsf{pk})$. Notice how we have reduced the number of calls to recrypt from N down to $n \cdot l$ and that we require only $n \cdot l$ constants v_{i_1,i_2} instead of the N constants w_i. The result is summarized in Algorithm 3. Assuming the $\left(\Gamma_{n,l}(0,\ldots,0,\psi^{i_1},0,\ldots,0) \right) \big|_\alpha$ and v_{i_1,i_2} are precomputed, the total cost of recrypting a ciphertext is essentially $n \cdot l \cdot T_{\mathsf{bits}} + 2 \cdot n \cdot l \cdot T_{\mathsf{mod},d}$.

Algorithm 3: Recrypting Ciphertext c version 2

$\overline{c} \leftarrow 0$.
for i_1 **from** 0 **upto** $n-1$ **do**
  for i_2 **from** 0 **upto** $l-1$ **do**
    $\overline{c}_{i_1,i_2} \leftarrow \mathsf{BitRecrypt}([c \cdot v_{i_1,i_2}]_d, \mathsf{pk})$.
    $\overline{c} \leftarrow \overline{c} \oplus \overline{c}_{i_1,i_2} \odot \left(\Gamma_{n,l}(0,\ldots,0,\psi^{i_1},0,\ldots,0) \right) \big|_\alpha$.
return $(\overline{c})$.

So far we have not exploited the SIMD capabilities of the somewhat homomorphic scheme. Therefore our next goal is to produce the recryptions $\overline{c}_{i_1,i_2}$ in parallel for $i_2 = 1,\ldots,l$. Thus we aim to compute a ciphertext $\hat{c}_{i_1}$ from c such that $\hat{c}_{i_1}$ represents a recryption of the message

$$(\mathsf{coeff}(\kappa_1, i_1), \ldots, \mathsf{coeff}(\kappa_l, i_1)) ,$$

where c represents an encryption of $(\kappa_1,\ldots,\kappa_l)$. We use the notation $\hat{c}_i$ to distinguish it from the recryption $\overline{c}_i$ above.

The key observation is that the recrypt procedure is the evaluation of a binary circuit, and that this binary circuit is identical (bar the constants) no matter which component we are recrypting. In addition the algebra splits into (at least) l finite fields of characteristic two, thus we can embed the binary circuit into each of these l components and perform the associated recryption in parallel. For a fixed i_1 we therefore want to execute the computation of the vector

$$([c \cdot v_{i_1,1} \cdot \zeta_0]_d \pmod{2}, \ldots, [c \cdot v_{i_1,l} \cdot \zeta_0]_d \pmod{2})$$

in the encrypted domain in parallel. Recall that each component of this vector is computed as

$$[c \cdot v_{i_1,k} \cdot \zeta_0]_d \pmod 2 = \bigoplus_{i=1}^{s} \bigoplus_{j=0}^{S-1} b_{i,j} \cdot y_{i,j}^{(k)} \pmod 2 \oplus \left\lceil \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot z_{i,j}^{(k)} \right\rfloor \pmod 2,$$

where $y_{i,j}^{(k)} = c \cdot v_{i_1,k} \cdot x_i \cdot R^j$ and $z_{i,j}^{(k)}$ an approximation of $y_{i,j}^{(k)}/d$ up to p bits after the binary point. Recall that to obtain the bit $\mathcal{B}_k = \left\lceil \sum_{i=1}^{s} \sum_{j=0}^{S-1} b_{i,j} \cdot z_{i,j}^{(k)} \right\rfloor \pmod 2$ we used the function $\mathsf{school\_book\_add}(M)$ with input an $s \times (p+1)$ array M where the ith row contained $\oplus_{j=0}^{S-1} b_{i,j} \cdot \mathsf{compute\_bits}(y_{i,j}^{(k)})$. In fact, $\mathcal{B}_k$ was simply the first bit in the bit vector returned by $\mathsf{school\_book\_add}(M)$.

If we now want to execute the above computation in the kth component (instead of the first), we basically have to multiply everything by $\Gamma_{n,l}(0,\ldots,0,1,0,\ldots,0)$, where $(0,\ldots,0,1,0,\ldots,0)$ is the vector of l elements of $\mathbb{K}_n$ whose kth element is equal to one, with all other elements being zero. To avoid costly modular multiplications by $\Gamma_{n,l}(0,\ldots,0,1,0,\ldots,0)\big|_\alpha$, we will use l different encryptions of $b_{i,j}$, depending on which of the l components of the algebra we are using. In particular, we no longer augment the public key with the data

$$\left(p, s, S, R, \{x_i, \{c_{i,j}\}_{j=0}^{S-1}\}_{i=1}^{s}\right),$$

where $c_{i,j} \leftarrow \mathsf{Encrypt}(b_{i,j}, \mathsf{pk})$, but instead replace the $c_{i,j}$ components with elements $e_{i,j,k}$ where

$$e_{i,j,k} \leftarrow \mathsf{Encrypt}\left(b_{i,j} \cdot \Gamma_{n,l}(0,\ldots,0,1,0,\ldots,0), \mathsf{pk}\right) \text{ for } 1 \le i \le s,\ 0 \le j < S,\ 0 \le k < l\,.$$

This means we need to increase the size of the augmented public key by essentially a factor of l. Once we have computed all the $\hat{c}_{i_1}$'s we can simply recover $\overline{c}$ by computing

$$\overline{c} \leftarrow \sum_{i_1=0}^{n-1} \hat{c}_{i_1} \odot \left(\left(\Gamma_{n,l}(\psi^{i_1},\ldots,\psi^{i_1})\right)\big|_\alpha\right).$$

The resulting algorithm is given in Algorithm 4. Note that to compute each $\hat{c}_{i_1}$ we only require one call to $\mathsf{school\_book\_add}(A)$ compared to l calls in Algorithm 3.

Algorithm 4: Recrypting Ciphertext c version 3: parallel recryption of all i_1th coefficients of the n elements embedded in a ciphertext c

```
c̄ ← 0.
for i_1 from 0 upto n − 1 do
    sum ← 0.
    A ← 0, where A ∈ M_{s×(p+1)}(Z/dZ).
    for i_2 from 0 upto l − 1 do
        c_{i_1,i_2} ← c · v_{i_1,i_2} (mod d).
        for j from 1 upto s do
            y ← c_{i_1,i_2} · x_j (mod d).
            for k from 0 upto S − 1 do
                if y is odd then
                    sum ← sum ⊕ e_{j,k,i_2}.
                b ← compute_bits(y).
                for u from 0 upto p do
                    A_{j,u} ← A_{j,u} ⊕ (b_u · e_{j,k,i_2}).
                y ← y · R (mod d).
    a ← school_book_add(A).
    ĉ_{i_1} ← sum ⊕ a_0.
    c̄ ← c̄ ⊕ ĉ_{i_1} ⊙ ((Γ_{n,l}(ψ^{i_1}, ..., ψ^{i_1})) |_α).
return (c̄).
```

We let $T_{\mathsf{par}}(n, l)$ denote the cost of performing this recryption operation on a message consisting of l field elements from $\mathbb{K}_n$ held in parallel. Assuming the $\left(\Gamma_{n,l}(\psi^{i_1}, \ldots, \psi^{i_1})\right)\big|_\alpha$ and the v_{i_1,i_2} are precomputed we obtain that

$$T_{\mathsf{par}}(n, l) = n\,(S \cdot s \cdot l + s \cdot l + l + 1) \cdot T_{\mathsf{mod},d} + n \cdot T_{\mathsf{school\_book\_add}}\,.$$

The main cost advantage therefore stems from the fewer calls to the function $\mathsf{school\_book\_add}$.

Naively it would appear that our parallel version of recrypt, using Algorithm 4, is more efficient than the naive version using Algorithm 2. However, one may need larger public keys to actually implement the parallel recryption (as it is a more complex circuit). We also need to compare whether doing operations in parallel and with large data entries (via the algebra A) is more efficient than doing the same operations but with bits using the standard bit-wise FHE scheme but with more complex circuits. It is to this topic we now turn by examining some "toy" examples.

6 Experimental Results

So the question arises as to whether it is simpler to perform FHE on bits, or to perform FHE via the algebra A. In this section we concentrate on estimating the performance in terms of the run time and the sizes of the resulting ciphertexts which need to be stored. First recall key generation; we choose N and a polynomial $F(X)$ with small coefficients, we then choose an element $\gamma \in \mathbb{Z}[\theta]$ which has coefficients of order 2^t. This results in a value for d of size approximately $N^N \cdot 2^{t \cdot N}$; thus we require roughly $t \cdot N$ bits to represent a single ciphertext.

We first let $T(n)$ denote the function which returns the number of $\mathbb{F}_2$ multiplications needed to perform a multiplication in the field $\mathbb{K}_n = \mathbb{F}_{2^n}$. Using Karatsuba multiplication (for example) we find, for n a power of two, that

$$T(n) := \begin{cases} 1 & \text{if } n = 1, \\ 3 \cdot T(n/2) & \text{otherwise.} \end{cases}$$

This is clearly only an estimate of the overall cost, as we are ignoring the required additions and management of the data.

There are various different options one has for implementing operations on l' finite fields each of size $2^{n'}$. In the following discussion we concentrate on the following four options; clearly other options are available but we select these as a way of demonstrating the different ways how our techniques could be used.

<u>Option 1:</u>: We operate on bits using the standard bit-wise FHE schemes, i.e. we take $n = l = 1$ in our FHE scheme. We will then require $l' \cdot n' \cdot t \cdot N$ bits to store our l' finite field elements, and the cost of performing a single SIMD style multiplication on the l' finite fields will cost around $l' \cdot T(n') \cdot T_{\mathsf{bits}}$ multiplications.

<u>Option 2:</u>: We operate on the l' finite field elements where each element uses a single ciphertext, i.e. we take $n = n'$ and $l = 1$ in our FHE scheme. This option has the benefit that we can work with the finite field, but we are not forced to operate in a SIMD manner all the time. With such an option we will require $l' \cdot t \cdot N$ bits to store our l' finite field elements, and performing a single SIMD style multiplication on the l' finite fields will cost around $l' \cdot T_{\mathsf{par}}(n', 1)$ multiplications.

<u>Option 3:</u>: We operate on all l' finite fields in a SIMD fashion using only a single ciphertext, i.e. we take $n = n'$ and $l = l'$ in our FHE scheme. Thus we will require $t \cdot N$ to store our l' finite field elements, and performing a single SIMD style multiplication on the l' finite fields will cost around $T_{\mathsf{par}}(n', l')$ multiplications.

<u>Option 4:</u>: Here we operate on bits, but we operate on them in a SIMD fashion by having a ciphertext represent l' bits, i.e. we take $n = 1$ and $l = l'$ in our FHE scheme. With this option we require $n' \cdot t \cdot N$ bits to store the l' finite field elements, and SIMD style multiplication will require $T(n') \cdot T_{\mathsf{par}}(1, l')$ multiplications.

We summarize the above choices, for the concrete parameters of $n' = 8$ and $l' = 16$, in the following table. We select a value for N around the size of 2000, purely to enable comparison with the work of [10]. We iterate this value is purely for illustrative purposes to show the difference between the various options; it should not be taken to indicate the $N \approx 2000$ is a secure security level. Fixing n', l' and N rather than leaving them variable is done as the overhead of the SIMD operations crucially depends on the specific combination of finite field and cyclotomic field chosen, and has no nice asymptotic meaning. We select a single parameter instance simply not to overwhelm the reader with data, since our goal is purely to show feasibility of our algorithms even at low security levels.

Note, that for Option 1 we select $N = 2048$ since if we are only encrypting bits then using the polynomial $F(X) = X^{2^n} + 1$ will always be more efficient than using $F(X) = \Phi_{3485}(X)$. In addition we keep the parameter t as an indeterminate, as we will be returning to that later.

	N	Ciphertext Space ($\approx$ bits)	Runtime Approx Cost
Option 1	2048	$262144 \cdot t$	$432 \cdot T_{\mathsf{bits}}$
Option 2	2560	$40960 \cdot t$	$16 \cdot T_{\mathsf{par}}(8, 1)$
Option 3	2560	$2560 \cdot t$	$T_{\mathsf{par}}(8, 16)$
Option 4	2560	$20480 \cdot t$	$27 \cdot T_{\mathsf{par}}(1, 16)$

Thus if one is soley interested in reducing the memory of the calculation one would select Option 3. To determine which one is most efficient one needs to actually implement the schemes, since the actual costs of each operation depend on the value of t needed. So we implemented the above algorithms for the four cases $(N, n, l) = (2048, 1, 1), (2560, 8, 1), (2560, 8, 16)$ and $(2560, 1, 16)$, so as to comparre the four options in the above analysis.

In all cases we found that taking $t = 400$ resulted in a scheme in which we were able to recrypt clean ciphertexts; however to enable fully homomorphic encryptions we need to recrypt dirty ciphertexts, and be able to perform some additional operations. For the first two of our four cases we found that $t = 600$ was sufficient, whilst for the second two we found that $t = 800$ was sufficient; note, we increased t in multiples of 100, thus smaller values could have been sufficient.

In the four cases we found the following recrypt times. We also present, assuming we wished in all cases to implement operations on $l' = 16$ values in $\mathbb{F}_{2^{n'}}$, where $n' = 8$, the actual time needed to perform the recrypt on such data and the total size of all ciphertexts needed to represent such data. In our implementation of the field algorithms for Option 1 and Option 4 we used the Karatsuba method mentioned above, and only performed recryption when implementing a multiplication using the FHE scheme; i.e. recryption was not performed upon additions. The algorithms were implemented in C++ using the NTL library and were run on a machine with six Intel Xeon 2.4 GHz processors and 47 GB of RAM.

Basic FHE Scheme				Performing Ops For $(n', l') = (8, 16)$		
(N, n, l)	t	(p, S)	Recrypt Time (sec)	Method	Recrypt Time (sec)	Ciphertext Size
$(2048, 1, 1)$	600	$(4, 32)$	15	Option 1	7148	18.00MB
$(2560, 8, 1)$	600	$(4, 32)$	187	Option 2	2983	3.00MB
$(2560, 8, 16)$	800	$(4, 32)$	735	Option 3	723	0.25MB
$(2560, 1, 16)$	800	$(4, 32)$	89	Option 4	2406	2.00MB

We end by noting the following: In our toy example we see that SIMD operations and parallel recryption offer some performance advantages. The exact benefit depends on a number of factors. Firstly the size of n' and l'; these are determined by an application and are often small. In turn n' and l' affect the choice of N, which also depends on the desired security level. The precise values of t and μ allowed are then determined by security analysis of lattice problems. Our toy experiments show that our ability to perform SIMD operations do not affect the size of t very much and that the parallel recryption operation is as practical as standard recryption.

The exact choice of which Option is best however depends on an application. Just as in standard SIMD vs non-SIMD operations on a standard processor, whether one utilizes the SIMD instructions in a program depends on the program being run.

7 Possible Applications

Before discussing two possible applications we note that one issue with SIMD operations on data is that sometimes we wish to move data between various elements in the l values on which we are operating. This is often a problem, since the hardware/mathematics/software which supports the SIMD operations precludes such operations. However, in our FHE scheme such operations can be performed at no additional cost.

Indeed given a SIMD word consisting of l elements in a finite field $\mathbb{F}_{2^n}$ one can produce a new SIMD word which consists of any linear function of the bits creating the original SIMD word. To see this we notice that it simply requires multiplying the matrix B used in the parallel recrypt procedure by the matrix defining the linear map. Thus, we can perform this linear function as part of the recryption performed for the previous operation.

In particular this means we can shuffle the elements in our SIMD word, or extract specific elements, or extract specific bits, etc. Indeed extracting specific bits in parallel was the core of our parallel recrypt procedure explained above.

We now turn to our two examples: The first example, namely homomorphic evaluation of AES under some homomorphic key, is used to demonstrate how SIMD operations in high level ($\mathbb{F}_{2^8}$) algebraic structures, allow us to evaluate complex operations relatively easily. Evaluation of AES circuits using FHE operations has been mentioned as a possible usage scenario in [13]. The second example, one of database lookup, provides an example of how data can be searched using SIMD style operations more efficiently than using the bit-wise homomorphic operations envisaged in [9].

In this section we assume that all operations are performed with post-processing by the recryption operation. Thus we are no longer interested in the size of the circuit which implements a functionality but simply the cost of the operations involved. As explained above we have essentially three key operations; the two algebraic operations Mult and Add, plus the linear operations on bits mentioned above. We shall denote the cost of these three operations by C_M, C_A and C_L, and we note that C_L essentially comes for free as part of recryption. For example, if an operation requires two multiplications, one addition and three linear operations we shall denote this cost (for simplicity) by $2 \cdot C_M + C_A + 3 \cdot C_L$.

7.1 Bit-Slicing

Any algorithm which is run on a circuit using bit operations can be run multiple times at once, by executing the algorithm on a set of parameters which supports operations on multiple bits in parallel. Such a technique is often called bit-slicing when applied to a single algorithm; however the technique is essentially also a bit-wise form of SIMD operation. Hence, *any* application performed using an FHE algorithm which supports the parallel recrypt procedure in this paper could be potentially sped-up by at least an order of magnitude by operating on multiple versions of the same algorithm in parallel.

7.2 Application to AES

As an example of the benefits of using FH-SIMD over the bitwise FHE we examine the case of how one would implement an AES functionality using FHE. Namely, we want a server to encrypt a message using a key which is only available via an FHE encryption. Using AES as a relatively complex example application of secure computation has also been recently suggested for a number of other related technologies; namely two and multi-party MPC [7, 14]. It is also particularly well suited to SIMD execution due to its overall design.

The method we propose is to encode the entire AES state matrix in a single ciphertext. Recall that the state matrix is a 4-by-4 matrix of elements in $\mathbb{F}_{2^8}$. We therefore first need to select an m so that the ideal (2) splits into at least 16 prime ideals of degree divisible by eight in the field defined by $\Phi_m(X)$. There are a large number of such examples, including the example we have used in this paper of taking $m = 3485$. Note that since $\phi(m)$ is equal to 4×16 we could also perform 4 AES computations in parallel as well, although we will restrict ourselves to one for ease of exposition. In terms of our previous section we let $K_8 = \mathbb{F}_{2^8}$ denote the standard representation of $\mathbb{F}_{2^8}$, i.e.

$$K_8 := \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1),$$

and we let A denote the algebra consisting of 64 copies of $\mathbb{F}_{2^{40}}$, each with the representation induced by the given factor of $\Phi_m(X) \pmod 2$.

We assume the AES state matrix is given by

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix},$$

which we encode as an element of K_8^{16} as $(s_{0,0}, s_{0,1}, \ldots, s_{3,3})$. Using the map $\Gamma_{8,16}$ we obtain an element of A, which can then be evaluated at α modulo p to obtain a trivial encryption of the message state (before the first round).

To implement AES we assume that the round keys k_i have been presented in encrypted form, using the above embedding via $\Gamma_{8,16}$. Computing the round keys from a given key can be done using the same operations needed to execute the rounds. Thus if we can implement the rounds using efficient FH-SIMD operations, then we can also compute the encryptions of the round keys given the initial key.

The round structure of AES is made up of four basic operations, which we now discuss in turn.

AddRoundKey This is the simplest operation and is clearly performed for all sixteen bytes in parallel by doing a single $\oplus$ operation of the FHE scheme. This step can be done at the cost of C_A.

ShiftRows In this operation row i is shifted left by $i-1$ positions. This is clearly an example of a *linear* operation from earlier, in that we map the ciphertext corresponding to

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, s_{1,0}, s_{1,1}, s_{1,2}, s_{1,3}, s_{2,0}, s_{2,1}, s_{2,2}, s_{2,3}, s_{3,0}, s_{3,1}, s_{3,2}, s_{3,3})$$

into a ciphertext corresponding to

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, s_{1,1}, s_{1,2}, s_{1,3}, s_{1,0}, s_{2,2}, s_{2,3}, s_{2,0}, s_{2,1}, s_{3,3}, s_{3,0}, s_{3,1}, s_{3,2}).$$

Since this is a reordering the cost is given by C_L.

MixColumns In this step we perform a matrix multiplication on the left of the state matrix by a fixed matrix given by

$$\begin{pmatrix} X & X+1 & 1 & 1 \\ 1 & X & X+1 & 1 \\ 1 & 1 & X & X+1 \\ X+1 & 1 & 1 & X \end{pmatrix}.$$

This is accomplished in four stages

1. Compute the trivial encryption c_1 of $\Gamma_{8,16}((X, X, \ldots, X))$, clearly this can be precomputed.
2. Compute $c_2 \leftarrow c \otimes c_1$.
3. By application of three *linear* operations we can create ciphertexts c_3, c_4, c_5 and c_6 corresponding to c_2 shifted up by one row, c shifted up by one row, c shifted up by two rows, and c shifted up by four rows (where shift rows is performed with rotation).
4. Compute $c_2 \oplus c_3 \oplus c_4 \oplus c_5 \oplus c_6$ and output the result.

Notice that our FH-SIMD scheme allows us to perform the 16 multiplications in parallel in the second step. The cost of the MixColumns operation is then $C_M + 4 \cdot C_A + 4 \cdot C_L$.

SubBytes This is the most complex of all the AES operations, however there is much existing literature on straight line (i.e. no branching) executions of the AES S-Boxes at byte level. For example the approach in [3] transforms the polynomial bases into a "nice" normal basis and then decomposes the arithmetic for inversion into $\mathbb{F}_{2^4}$ and then $\mathbb{F}_{2^2}$ operations. At which point all the arithmetic is just logical operations, and hence amenable to FH-SIMD operations. However, this approach is more suited to real hardware, or to FH-SIMD operations where the basic data type is a bit (e.g. when using say $(n, l) = (1, 16)$ in our main scheme).

As we are restricted to operations which can be performed efficiently in our FH-SIMD scheme a more naive approach is probably to be preferred. Recall that the AES S-Box consists of inverting each state byte in K_8 (where we define $0^{-1} = 0$), followed by an $\mathbb{F}_2$-linear operation. Also recall that $x^{-1} = x^{254}$ in the field K_8. We can therefore apply the S-Box operation to our encrypted state using the following method:

- $t \leftarrow c$.
- For $i = 1$ to 6 do
  - $t \leftarrow t \otimes t$.
  - $t \leftarrow t \otimes c$.
- $t \leftarrow t \otimes t$.
- Extract eight ciphertexts $t_0, \ldots, t_7$ such that t_i is the (parallel) encryption of the i-th bit of all 16 values in t.
- Perform the linear operation on $t_0, \ldots, t_7$ in parallel to produce ciphertexts $s_0, \ldots, s_7$.
- Map these ciphertexts back to an encryption of an element in A.

The first step, that of producing an encryption t of x^{254} where c is an encryption of x, requires at most 13 fully homomorphic multiplications. The second step of extracting the ciphertexts $t_0, \ldots, t_7$ is essentially a single linear operation. The third step of adding the elements $t_0, \ldots, t_7$ together to produce $s_0, \ldots, s_7$, requires $4 \cdot 8 = 32$ homomorphic additions, due to the nature of the linear operation in AES. The final step of obtaining a single ciphertext from $s_0, \ldots, s_7$ is also an application of a linear operation. Thus the total cost of SubBytes is given by $13 \cdot C_M + 32 \cdot C_A + 2 \cdot C_L$.

We note that our SIMD evaluation of the AES round function not only benefits in our system from being able to execute 16 operations in parallel. We also have the benefit of being able to deal directly with $\mathbb{F}_{2^8}$ arithmetic operations, as well as decompose into bits where necessary in the linear transformation in the S-Box operation. The total cost of a round function being given by

$$14 \cdot C_M + 37 \cdot C_A + 7 \cdot C_L,$$

although by interleaving operations a lower cost could probably be obtained.

7.3 Data Base Lookup

We end by examining a more realistic application scenario, namely one of searching an encrypted database on a remote server. Suppose a user has previously encrypted a database and stored it on a cloud service provider, and now she wishes to retrieve some of the data. We first note that the usual atomic database operation of search actually consists of two operations. The first operation is one of search, whereas the second is one of retrieval. The following method performs the search using FHE and the retrieval using Private Information Retrieval (PIR).

We assume the database is such that one can determine beforehand which fields will be searched on. In some sense this is akin to the basic premise of public key encryption with keyword search [1], however we have a more complicated data retrieval operation to perform. To simplify the discussion we assume that there is only one database field which is searchable, and another field which contains the information. Each database entry (in the clear) is then given by a tuple (i, s, d), where s is the search term, d is the data and i is some index which is going to enable retrieval. The number of such items we denote by r. We assume that i and s are n bits in length, and thus can be encoded as an element of the finite field $K_n = \mathbb{F}_{2^n}$.

To encrypt the database the user picks a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ for our FH-SIMD scheme, as well as a symmetric key K for a symmetric encryption scheme (E_K, D_K). Let us assume that the encryption scheme can support l operations in $\mathbb{F}_{2^n}$ in parallel. When placing the database on the cloud service provider the user divides the database into $\lceil r/l \rceil$ blocks of l items. Then to actually send the server the jth encrypted data block, for $j = 0, 1, 2, \ldots, \lceil r/l \rceil - 1$ we send

$$\begin{aligned}(\mathbf{i}_j, c_j, \mathbf{E}_j) = \big(&i_{l \cdot j+1}, \ldots, i_{l \cdot (j+1)}, \\ &\mathsf{Encrypt}(\Gamma_{n,l}(s_{l \cdot j+1}, \ldots, s_{l \cdot (j+1)}), \mathsf{pk}), \\ &E_K(d_{l \cdot j+1}), \ldots, E_K(d_{l \cdot (j+1)})\big) .\end{aligned}$$

We now discuss how the user retrieves all data items which correspond to the search term s. We first recover an encryption of an encoding of the index terms which contain this search term. This is done by sending the server one ciphertext, and receiving one in return. The sent "query" ciphertext is equal to

$$q = \mathsf{Encrypt}(\Gamma_{n,l}(s, \ldots, s), \mathsf{pk}),$$

i.e. an encryption of l copies of the query term s.

The server then takes each data block $(\mathbf{i}_j, c_j, \mathbf{E}_j)$ and computes $c_j^{(1)} = q \oplus_{\mathsf{pk}} c_j$. The value $c_j^{(1)}$ is then homomorphically raised to the power $2^n - 1$, by performing $2n$ applications of Mult. This results in a ciphertext $c_j^{(2)}$ which is an encryption of a vector of zero and ones, with a one only occurring in position k when s is not equal to the kth component of the vector underlying the ciphertext c_j.

The server then computes $c_j^{(3)} = (c_j^{(2)} \oplus_{\mathsf{pk}} \mathsf{Encrypt}(\Gamma_{n,l}(1, 1, \ldots, 1), \mathsf{pk})) \otimes_{\mathsf{pk}} \mathsf{Encrypt}(\Gamma_{n,l}(\mathbf{i}_j), \mathsf{pk})$, and the set of ciphertexts $c_j^{(3)}$ are then added together using Add to obtain a final ciphertext c', which is returned to the user. Note, that this "search" query has a cost of $(2 \cdot n + 1) \cdot C_M + 2 \cdot C_A$ per data block.

The plaintext underlying the returned ciphertext c' consists of l components, where the kth component is given by

$$\bigoplus_{s=s_{l\cdot j+k}} i_{l\cdot j+k}.$$

If there is only one match per component then we have recovered the matching indices and hence can recover the actual data by engaging in a PIR protocol [4, 12]. The problem arises when we have the possibility of more than one match per component per query. In this situation we need an encoding algorithm to enable us to recover the exact PIR inputs we need to recover the data.

In the extreme case we have a possibility of every component containing $\lceil r/l \rceil$ matches, i.e. the search term s matches with every item in the database. In which case we obtain, via a trivial encoding, that we must have $\lceil r/l \rceil \le n$. This essentially implies that the length of the database is bounded by the number of bits we can encrypt, i.e. $r < l \cdot n$.

However, if we can ensure that a maximum of t matches can occur per SIMD component then we can produce a more effective encoding as follows: Firstly we assume the encoding used for data retrieval in the PIR is such that we recover the data item corresponding to an index/component position pair. This simplifies our discussion as we only have to concentrate on decoding a single component.

We set $m = \lceil r/l \rceil$, and to each of the m blocks we associate an n-bit index i. We want to therefore be able, given an xor of the indices $z = i_{j_1} \oplus \ldots \oplus i_{j_s}$, with $s \le t$, to recover the set $\{i_{j_1}, \ldots, i_{j_s}\}$. To construct the encoding we take the parity matrix of an $[N, K, D]$ linear code over $\mathbb{F}_2$ of length N, rank K and minimum distance D, which we assume is greater $2 \cdot t$. This is a matrix of dimension $(N - K) \times N$. We then take as our indices the columns of this matrix, which implies that these indices must fit in n bits, hence $N - K \le n$. Given an xor of at most t indices we can recover which indices were xor-ed together by decoding the $[N, K, D]$ linear code. To see this notice that the sum of indices z is a syndrome of a codeword in the linear code. Thus by recovering the error positions in the code from the syndrome we know which indices, i.e. which columns of the parity check matrix, were xor-ed together. Thus the total number of distinct indices we can cope with is bounded by the column size of the parity check matrix, i.e. N. Hence, we obtain $m = \lceil r/l \rceil \le N$.

As an example of a possible encoding scheme we take a primitive BCH code which exists for any pair of values of (s, t) such that $s \ge 3$ and $t < 2^{s-1}$. The primitive BCH code over $\mathbb{F}_2$ then has parameters given by $N = 2^s - 1$, $N - K \le s \cdot t$ and $D \ge 2 \cdot t + 1$. If we take our FHE scheme of the previous section using the mth cyclotomic polynomial with $m = 3485$, then we have $l = 64$, $n \le d = 40$ and $\phi(m) = 2560$. Given the bounds

$$\lceil r/l \rceil \le N = 2^s - 1 \text{ and } s \cdot t \le n,$$

and supposing we take $t = 3$, so we can recover at most three collisions on search terms within each component, then by setting $n = d = 40$ and $(s, t) = (13, 3)$ we obtain a valid encoding. This implies that the total number of items within the database is bounded by $l \cdot N = 524224$. Clearly using more optimal codes, or different cyclotomic polynomials one can obtain larger values of the whole database, or one can deal with more collisions within a component.

The above methodology using our FH-SIMD scheme to search on l components at once in an efficient manner, results in a linear speed up in the search of the encrypted database. However, there is another advantage of our splitting the database into l components; we can deal with (albeit having a probability of invalid indices being returned) having more collisions between the search terms. In the above example we could deal with up to three collisions in each component, this meant that our method would be guaranteed to be correct if there were at most three items in the database corresponding to each search item. However, if we assume that the search items are randomly distributed between the l components, then in practice we can deal with more collisions, since our results will be correct as long as there are at most t collisions *per component*. The generalised birthday bound [17] says that we can have

$$(t!)^{1/t} \cdot l^{(t-1)/t}$$

collisions before the probability of obtaining more than t collisions in one of the l components is greater than $1/2$. In our above numerical example, with $t = 3$ and $l = 64$, this equates to just over 29 matches in our database.

8 Acknowledgements

This material is based on research sponsored by the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II. The first author was also supported by the Defense Advanced Research Projects

Agency (DARPA) and Air Force Research Laboratory (AFRL) under agreement number FA8750-11-2-0079, by the Royal Society via a Royal Society Wolfson Merit Award, by the ERC via an Advanced Grant, and the EPSRC via grant EP/I03126X. The second author was supported by a Postdoctoral Fellowship of the Research Foundation - Flanders (FWO).

References

1. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. *Advances in Cryptology – Eurocrypt 2004*, Lecture Notes in Comput. Sci. **3027**, 506–522, 2004.
2. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. To appear *Advances in Cryptology – Crypto 2011*, Lecture Notes in Comput. Sci. **XXXX**, XXXX–XXXX, 2011.
3. D. Canright. A very compact S-Box for AES. *Cryptographic Hardware and Embedded Systems – CHES 2005*, Lecture Notes in Comput. Sci. **3659**, 441–455, 2005.
4. B. Chor, E. Kushilevitz, O. Goldreich and M. Sudan. Private information retrieval. *J. ACM*, **45**, 965–981, 1998.
5. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. *Advances in Cryptology – Eurocrypt 2010*, Lecture Notes in Comput. Sci. **6110**, 24–43, 2010.
6. J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, **19**, 297–301, 1965.
7. I. Damgård and M. Keller. Secure multiparty AES. *Financial Cryptography – FC 2010*, Lecture Notes in Comput. Sci. **6052**, 367–374, 2010.
8. C. Gentry. Fully homomorphic encryption using ideal lattices. *Symposium on Theory of Computing – STOC 2009*, ACM, 169–178, 2009.
9. C. Gentry. A fully homomorphic encryption scheme. *Manuscript*, 2009.
10. C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. *Advances in Cryptology – Eurocrypt 2011*, Lecture Notes in Comput. Sci. **6632**, 129–148, 2011.
11. I.J. Good. The interaction algorithm and practical Fourier analysis. *J.R. Stat. Soc.*, **20**, 361–372, 1958.
12. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. *Foundations of Computer Science – FoCS '97*, 364–373, 1997.
13. K. Lauter, M. Naehrig, V. Vaikuntanathan. Can homomorphic encryption be practical. Preprint, 2011.
14. B. Pinkas, T. Schneider, N.P. Smart, S.C. Williams. Secure two-party computation is practical. *Advances in Cryptology – Asiacrypt 2009*, Lecture Notes in Comput. Sci. **5912**, 250-267, 2009.
15. C.M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proc. IEEE*, **56**, 1107–1108, 1968.
16. N.P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. *Public Key Cryptography – PKC 2010*, Lecture Notes in Comput. Sci. **6056**, 420–443, 2010.
17. K. Suzuki, D. Tonien, K. Kurosawa and K. Toyota. Birthday paradox for multi-collisions. *Information Security and Cryptology – ICISC 2006*, Lecture Notes in Comput. Sci. **4296**, 29–40, 2006.
18. L.H. Thomas. Using a computer to solve problems in physics. *Application of Digital Computers*, 1963.