Improved Key Generation For Gentry's Fully Homomorphic Encryption Scheme

P. Scholl and N.P. Smart

Dept. Computer Science,
University of Bristol,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.

Abstract. A key problem with the original implementation of the Gentry Fully Homomorphic Encryption scheme was the slow key generation process. Gentry and Halevi provided a fast technique for 2-power cyclotomic fields. We present an extension of the Gentry–Halevi key generation technique for arbitrary cyclotomic fields. Our new method is roughly twice as efficient as the previous best methods. Our estimates are backed up with experimental data.

The major theoretical cryptographic advance in the last three years was the discovery by Gentry in 2009 of a fully homomorphic encryption scheme [4, 5]. Gentry's scheme was initially presented as a completely theoretical construction, however it was soon realised that by specialising the construction one could actually obtain a system which could at least be implemented; although not yet in such a way as to enable practical computations. The first such implementation was presented by Smart and Vercauteren [10]. The Smart and Vercauteren implementation used arithmetic of cyclotomic number fields. In particular they focused on the field generated by the polynomial $F(X) = X^{2^n} + 1$, but they noted that the scheme could be applied with arbitrary (even non-cyclotomic) number fields. A main problem with the version of Smart and Vercauteren was that the key generation method was very slow indeed.

In [6] Gentry and Halevi presented a new implementation of the variant of Smart and Vercauteren, but with a greatly improved key generation phase. In particular Gentry and Halevi note that key generation (for cyclotomic fields) is essentially an application of a Discrete Fourier Transform, followed by a small amount of computation, and then application of the inverse Discrete Fourier Transform. They then show that one does not even need to perform the DFT's if one selects the cyclotomic field to be of the form $X^{2^n} + 1$. They do this by providing a recursive method to deduce two constants, from the secret key, which enables the key generation algorithm to construct a valid associate public key. The key generation method of Gentry and Halevi is fast, but appears particularly tailored to working with two-power roots of unity.

However, the extra speed of their key generation method comes at a cost. Restricting to two-power roots of unity means that one is precluded from the

type of SIMD operations discussed in [11]. To enable such operations one needs to be able to deal with general cyclotomic number fields. In [11] it is pointed out that the DFT/inverse-DFT method can be easily applied to the case of general cyclotomic fields via the use of the FFT algorithms such as those of Good–Thomas [7, 12], Rader [9] and others. However, the simple recursive method of Gentry and Halevi does not seem to apply.

Other works have examined ways of improving key generation, and fully homomorphic encryption schemes in particular. For example [8] has a method to construct keys for essentially random number fields by pulling random elements and analyzing eigenvalues of the corresponding matrices; this method however does not allow the efficiency improvements of [10] and [6] with respect to reduced ciphertext sizes etc. More recent fully homomorphic schemes based on the LWE assumption [3] have more efficient key generation procedures than the original Gentry scheme; and appear to be more suitable in practice. However for this work we concentrate purely on the schemes in the "Gentry family".

In this paper we present an analysis of the key generation algorithm, for Gentry based schemes, for general cyclotomic fields, generated by the the primitive m-th roots of unity. In particular, we show that Gentry and Halevi's recursive method can be generalised to deal with prime power values of m, and also any m with just a few small, repeated prime factors. We also show for general m that the DFT/inverse-DFT method is sub-optimal, and that an algorithm exists which requires only a single DFT application to compute the secret key. Our general key generation method is essentially twice as fast as previous methods; both theoretically and in practice.

The paper is organized as follows: In Section 1 we present the required mathematical background and notation. In Section 2 we present the required information about the key generation method for the variant of Gentry's scheme we will be discussing. Then in Section 3 we describe how one could execute the key generation procedure assuming as soon as two coefficients of one associated polynomial $g(X)$ and one coefficient of another associated polynomial $h(X)$ are computed. Algorithms to compute these three coefficients are then presented in Section 4. Finally in Section 5 we present some experimental results.

1 Mathematical Background

Let $F(X) = \Phi_m(X)$ denote the m-th cyclotomic polynomial, i.e. the irreducible polynomial whose roots are the primitive m-th roots of unity. This polynomial has degree $N = \phi(m)$, where $\phi(\cdot)$ is Euler's phi-function. We let the m-th roots of unity be denoted by $\omega_m^0, \ldots, \omega_m^{m-1}$, which are defined as powers of $\omega_m = \exp(\frac{2\pi\sqrt{-1}}{m})$, the principal m-th root of unity. The roots of $F(X)$ are those values ω_m^i where $\gcd(i, m) = 1$. We let $\rho_0, \ldots, \rho_{N-1}$ denote these primitive m-th roots of unity (i.e. the roots of F).

If $f(X) \in \mathbb{Z}[X]$ is an arbitrary polynomial then we let f_i denote the coefficient of X^i in $f(X)$. For a polynomial $f(X)$ we let $\|f\|_\infty = \max_{i=0}^{\deg(f)} |f_i|$ denote the infinity-norm (i.e. the max-norm) of its coefficient vector. Given two polynomials

$f(X)$ and $g(X)$ the resultant of f and g is defined to be

$$\mathsf{resultant}(f,g) = \prod_{\alpha,\beta}(\alpha - \beta)$$

where α ranges over the roots of $f(X)$ and β ranges over the roots of $g(X)$. We also have that

$$\mathsf{resultant}(f,g) = \prod_{\alpha} g(\alpha). \tag{1}$$

Given a polynomial $x(X)$ of degree $m-1$, which is simply a list of coefficients $x_0, x_1, \ldots, x_{m-1}$, the Discrete Fourier Transform (DFT) is defined by the evaluation of this polynomial at all of the m-th roots of unity. So the k-th coefficient of the DFT is then

$$\mathbf{x}_k = \sum_{i=0}^{m-1} x_i \omega_m^{i \cdot k}.$$

Naïve computation of the DFT from this definition takes $O(m^2)$ operations. Fast Fourier Transform (FFT) algorithms reduce this to $O(m \log m)$. The inverse-DFT is the procedure which takes m evaluations of a polynomial at the m-th roots of unity, and then recovers the polynomial. We write $\mathbf{x} \leftarrow \mathsf{DFT}(x)$ and $x \leftarrow \mathsf{DFT}^{-1}(\mathbf{x})$.

2 Key Generation for Gentry

Key generation for Gentry's FHE scheme depends on two parameters m and t. The value m defines the underlying cyclotomic field as above, and we define $N = \phi(m)$, which is the degree of the cyclotomic polynomial $F(X)$. The parameter t is used to define how "small" the secret key is. Note that in practice the word "small" is a relative term and we are not really dealing with small numbers at all. To generate keys for Gentry's FHE scheme one can proceed as follows:

- $v(X) \leftarrow \mathbb{Z}[X]$ with $\|v\|_\infty \leq 2^t$ and $v(X) \equiv 1 \pmod 2$.
- Compute $w(X) \in \mathbb{Z}[X]$ such that

$$d = v(X) \cdot w(X) \pmod{F(X)}$$

  where $d = \mathsf{resultant}(v, f)$.
- If $v(X)$ and $w(X)$ do not have a common root modulo d then return to the beginning and choose another $v(X)$.
- Let $\alpha \in \mathbb{Z}_d$ denote the common root.
- Set $\mathfrak{pk} \leftarrow (\alpha, d)$ and $\mathfrak{sk} \leftarrow (w(X), d)$.

Note, there are various minor variations on the above procedure in the literature. In Smart and Vercauteren [10] the polynomial $v(X)$ is rejected unless d is prime; this is done due to the method the authors used to compute the common root α. Gentry and Halevi [6] notice that if $v(X)$ and $f(X)$ have a common root modulo $f(X)$ then it is given by $\alpha = -w_{N-1}/w_0 \pmod d$. Gentry and Halevi, make an

additional modification, in that the condition on $v(X) \equiv 1 \pmod 2$ is dropped, and replaced by the condition that $d \equiv 1 \pmod 2$; this means the authors only need to compute one coefficient of $w(X)$ for their application. However, in [11], the authors show that selecting $v(X) \equiv 1 \pmod 2$ enables SIMD style operations on data, as long as $m \neq 2^r$. They also show that whilst all coefficients of $w(X)$ are needed in the secret key, one can generate all of them via the relation

$$w_i = \begin{cases} \alpha w_{i+1} + F_{i+1} w_{N-1} & \pmod d \quad \text{if } 0 \leq i < N-1 \\ -\alpha w_0 & \pmod d \quad \text{if } i = N-1 \end{cases} \tag{2}$$

The main question is then how to compute w_0 and d. In [6, 11] it is pointed out that the following DFT-based procedure can be applied:

- $\mathbf{v} \leftarrow \mathsf{DFT}(v(X))$.
- $d \leftarrow \prod_{\gcd(i,m)=1} \mathbf{v}_i$.
- $\mathbf{w}_i \leftarrow d/\mathbf{v}_i$.
- $w(X) \leftarrow \mathsf{DFT}^{-1}(\mathbf{w})$.

Gentry and Halevi [6] then go on to notice that one can actually compute $w(X)$ and d without any need for computing DFTs. They do this, since they solely focus on the case $m = 2^r$, which enables them to present the calculation of d and $w(X)$ as the calculation of computing two coefficients of an associated polynomial $g(X)$.

In this paper we generalise this method of Gentry and Halevi to arbitrary values of m; for non-prime powers of m we will still require the application of a single DFT algorithm, but will no longer need the inverse DFT. The key observation is that d and $w(X)$ are related, for general m, to the coefficients of *two* associated polynomials $g(X)$ and $h(X)$. It is to these polynomials, and their properties, that we now turn.

3 The Polynomials $g(X)$ and $h(X)$

Before proceeding we introduce Ramanujan sums, for those readers who are not acquainted with them. A Ramanujan sum is simply a sum of powers of primitive roots of unity:

$$C_m(k) := \sum_{\substack{i=0 \\ (i,m)=1}}^{m-1} \omega_i^k = \sum_{d|(k,m)} \mu\left(\frac{m}{d}\right) d$$

where the second sum is over the positive divisors of $\gcd(k, m)$, and μ is the Möbius function. For a proof of this formula see e.g. [2, p. 162]. The Ramanujan sum can therefore be easily computed provided m can be factored efficiently; this will always be the case in our applications since m is a small integer. It is clear from this formula that $C_m(-k) = C_m(k)$. We also have the following result, which we will need:

Proposition 1. *Let F_i denote the i-th coefficient of the m-th cyclotomic polynomial $F(X)$. Then for $k = 0, \ldots, N-1$,*

$$\sum_{i=1}^{N-1} C_m(i-k) \cdot F_{i+1} = -C_m(-k-1).$$

Proof. Suppose that θ is a root of F. Observing that since F is a cyclotomic polynomial, $F_0 = F_N = 1$, and so

$$-1 = \sum_{i=1}^{N} F_i \theta^i = \sum_{i=0}^{N-1} F_{i+1} \theta^{i+1}.$$

This is equivalent to

$$-\theta^{-k-1} = \sum_{i=0}^{N-1} F_{i+1} \theta^{i-k}.$$

The above relation can then be applied to the individual summands in $C_m(k)$ (which are powers of the roots of F) to give the desired result.

We now turn to our key generation method. Given $v(X)$ we define the following polynomials,

$$g(X) := \prod_{i=0}^{N-1} (v(\rho_i) - X)$$
$$h(X) := \prod_{i=0}^{N-1} (v(\rho_i) - X/\rho_i).$$

The polynomial g here is the same as that defined in [6]. However, when m is not a power of 2 we also need to introduce $h(X)$ in order to help us find w.

The constant-term and degree one coefficients of these polynomials, i.e. g_0, g_1, h_0 and h_1, must then be computed. We leave discussion of how this step is done until the next section. In this section we detail how, given these coefficients, we can compute $w(X)$ and d. Note that because of Equation 1, the values g_0 and h_0 are both equal to the resultant, d, of v and f.

We also have

$$g_1 = -\sum_{i=0}^{N-1} \prod_{j \neq i} v(\rho_j) = -\sum_{i=0}^{N-1} \frac{\prod_{j=0}^{N-1} v(\rho_j)}{v(\rho_i)} = -\sum_{i=0}^{N-1} \frac{d}{v(\rho_i)} = -\sum_{i=0}^{N-1} w(\rho_i) \quad (3)$$

and similarly,

$$h_1 = -\sum_{i=0}^{N-1} \frac{w(\rho_i)}{\rho_i}. \quad (4)$$

To determine the coefficients of w, we first look at a more general form of the above expressions for g_1 and h_1, and show how this relates to w. Define for $k \geq 0$ the following sequence of sums

$$W_k := \sum_{i=0}^{N-1} \frac{w(\rho_i)}{\rho_i^k}.$$

Our strategy from here onwards is to give a simple expression for W_k in terms of the coefficients of w, and then show that the values of W_k can be easily computed independently using the information we already have of g_1 and h_1. Next, by looking at successive terms of W_k, a set of simultaneous equations involving the coefficients of w will arise, and it will be shown that these can be solved to recover all of w.

Observe that, as a result of Equations 3 and 4, we have $W_0 = -g_1, W_1 = -h_1$. More generally, we see that

$$W_k = \sum_{i=0}^{N-1} \frac{\sum_{j=0}^{N-1} w_j \cdot \rho_i^j}{\rho_i^k} = \sum_{j=0}^{N-1} w_j \cdot \sum_{i=0}^{N-1} \rho_i^{j-k} = \sum_{j=0}^{N-1} C_m(j-k) \cdot w_j.$$

Thus the above equation gives us an expression for W_k as a simple linear combination of the coefficients of w, by the Ramunujan sums $C_m(j-k)$. Applying Equation 2, this allows us to deduce

Proposition 2.

$$W_k = \alpha \cdot W_{k+1} \pmod{d}.$$

Proof.

$$\begin{aligned}
W_k &= \sum_{i=0}^{N-1} C_m(i-k) \cdot w_i \\
&= \sum_{i=0}^{N-2} C_m(i-k) \cdot \alpha \cdot w_{i+1} + w_{N-1} \cdot \sum_{i=0}^{N-2} C_m(i-k) \cdot F_{i+1} \\
&\quad + C_m(N-k-1) \cdot w_{N-1} \\
&= \alpha \cdot \sum_{i=0}^{N-2} C_m(i-k) \cdot w_{i+1} + w_{N-1} \cdot \sum_{i=0}^{N-1} C_m(i-k) \cdot F_{i+1} \\
&= \alpha \cdot \sum_{i=1}^{N-1} C_m(i-k-1) \cdot w_i - w_{N-1} \cdot C_m(-k-1) \\
&= \alpha \cdot \sum_{i=1}^{N-1} C_m(i-k-1) \cdot w_i + \alpha \cdot w_0 \cdot C_m(-k-1) \\
&= \alpha \cdot W_{k+1}
\end{aligned}$$

From which comes the following immediate corollary:

Corollary 1.

$$W_k = -g_1 \cdot \alpha^{-k} \pmod{d}.$$

Note that Proposition 2 immediately implies that $\alpha = g_1/h_1 \bmod d$, and thus any value of W_k can be easily determined using the corollary. This allows us to create a system of linear equations in the coefficients of w, from the values of $W_0, \ldots, W_{N-1}$, as follows:

$$\begin{pmatrix} C_m(0) & C_m(1) & \cdots & C_m(N-1) \\ C_m(1) & C_m(0) & \cdots & C_m(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ C_m(N-2) & C_m(N-3) & \cdots & C_m(1) \\ C_m(N-1) & C_m(N-2) & \cdots & C_m(0) \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{pmatrix}$$

$$= -g_1 \cdot \begin{pmatrix} 1 \\ \alpha^{-1} \\ \alpha^{-2} \\ \vdots \\ \alpha^{1-N} \end{pmatrix} \pmod{d}$$

We write the above equation as $C \cdot \boldsymbol{w} = -g_1 \cdot \boldsymbol{\alpha}$. The matrix C possesses the interesting property that every diagonal is constant; as such it is a symmetric Toeplitz matrix. There is a method to solve such a system of equations in only $O(N^2)$ operations, as opposed to the usual $O(N^3)$ required for a general matrix [13]. We note, that for a given value of m the matrix C is fixed and hence computing its inverse can be considered as a precomputation. Thus with this precomputation the cost of computing the key, given the coefficients g_0, g_1, and h_0, is a linear operation in N.

When it comes to computing the inverse of the matrix C, we note that it appears experimentally to be of the form, for all m,

$$C^{-1} = \frac{1}{m} Z,$$

for some integral $N \times N$ matrix Z whose coefficients are bounded in absolute value by m. However, we were unable to prove this. In any case we can assume this is true, then efficiently compute the inverse of C by inverting C/m using standard floating point arithmetic and then rounding the resulting coefficients to integers. This matrix can then be divided by m, tested for correctness and stored.

4 Determining g_0, g_1 and h_1

In this section we examine methods to determine the coefficients g_0, g_1 and h_1. We first present a general method, which works for arbitrary values of m and

leads to key generation that is essentially twice as fast as existing methods. We then describe a method for "special" values of m, namely those containing a large number of repeated factors, such as when m is a prime power. By specialising the results of this section, and the method in the previous section to the case $m = 2^r$, we obtain the key generation method of Gentry and Halevi.

4.1 General m

We note that the desired coefficients of g and h can be computed directly from the FFT of v. Thus by applying one FFT and the techniques of the previous section we can avoid the second inverse-FFT required of the method in Section 2. Hence, we can obtain a method which is essentially twice as fast as that proposed in 2.

Recall that the FFT of v gives the values $v(\rho_0), v(\rho_1), \ldots, v(\rho_{N-1})$. With these computed, g_0 is obtained by simply multiplying them together (as is done in the FFT-based key generation algorithm). Then note that

$$g_1 = -\sum_{i=0}^{N-1} \frac{g_0}{v(\rho_i)}$$

and

$$h_1 = -\sum_{i=0}^{N-1} \frac{g_0}{\rho_i \cdot v(\rho_i)}.$$

So the coefficients g_1 and h_1 can all be computed in $O(N)$ operations (albeit on numbers of $O(N \cdot t)$ bits in length), once the initial FFT of v is computed. This may not seem a major improvement, after all we have only really saved one FFT out of two; but there is a huge implied constant in the big-Oh notation due to the fact that the coefficients of the polynomial $w(X)$ are all of size around $2^{N \cdot t}$, which is practice will result in many millions of bits of precision being needed in the FFT algorithms.

4.2 The case $m = p^r$

We first define the following two polynomials

$$a(X) = \prod_{j=0}^{p-1} v(\alpha_j \cdot X)$$

$$b(X) = \sum_{j=0}^{p-1} \prod_{j \neq i} v(\alpha_j \cdot X).$$

where $\alpha_0, \ldots, \alpha_{p-1}$ denote the p-th roots of unity. By elementary Galois theory we find that the coefficients of a must be rational integers. We observe that $a(\alpha_i \cdot X) = a(X)$, so it must follow that the i-th coefficient of a will be zero if i

is not a multiple of p. By a similar argument we also deduce that $b(X) \in \mathbb{Z}[X]$ and that $b_i = 0$ if i is not a multiple of p.

Our algorithm will depend on starting with the polynomials $a(X)$ and $b(X)$. These can be easily computed due to the following observations. Firstly, by [1, Proposition 4.3.4], we have

$$a(X^p) = p^{1-p} \cdot \mathsf{resultant}_Y(v(Y), p \cdot X - p \cdot Y^p).$$

where $\mathsf{resultant}_Y(f, g)$ denotes the resultant polynomial in Y of the bivariate polynomials f and g. Note that when computing this resultant, every occurrence of Y^p in the polynomial $v(Y)$ can be replaced with X to vastly speed up computation time.

Now notice also that

$$b(X) = \sum_{i=0}^{p-1} \frac{a(X)}{v(\alpha_i \cdot X)} = \sum_{i=0}^{p-1} \frac{a(\alpha_i \cdot X)}{v(\alpha_i \cdot X)} = \sum_{i=0}^{p-1} (a/v) \cdot (\alpha_i X).$$

Then by writing $(a/v)(X) = \sum_{j=0}^{N-1} B_j \cdot X^j$ and changing the order of summations, we obtain:

$$b(X) = \sum_{j=0}^{N-1} B_j \cdot X^j \cdot \left(\sum_{i=0}^{p-1} \alpha_i^j \right) = p \sum_{j=0}^{N/p-1} B_{p \cdot j}.$$

So the polynomial $b(x)$ can be computed from the coefficients of the quotient polynomial a/v; note that this is an exact polynomial division over $\mathbb{Z}[X]$.

Now recall the definition of g, in terms of v evaluated at the primitive roots of unity:

$$g(X) := \prod_{i=0}^{N-1} (v(\rho_i) - X).$$

Since $m = p^r$, it can be shown that the primitive m-th roots of unity are heavily related to the p-th roots of unity, $\alpha_0, \ldots, \alpha_{p-1}$. For any $k \in \{0, \ldots, p-1\}$,

$$\rho_{i+k \cdot N/p} = \alpha_k \cdot \rho_i.$$

Using this fact, the length-$(N-1)$ product defining g above can be re-expressed as a length-$(N/p-1)$ product of p-products, involving the p-th roots of unity. Applying this to g and then evaluating modulo X^2 (to obtain the lowest two

coefficients) gives

$$\begin{aligned} g(X) &= \prod_{i=0}^{N/p-1} \prod_{j=0}^{p-1} (v(\alpha_j \cdot \rho_i) - X) \\ &= \prod_{i=0}^{N/p-1} \left(\underbrace{\prod_{j=0}^{p-1} v(\alpha_j \cdot \rho_i)}_{a(\rho_i)} - X \cdot \underbrace{\sum_{j=0}^{p-1} \prod_{j \neq i} v(\alpha_j \cdot \rho_i)}_{b(\rho_i)} \right) \pmod{X^2} \\ &= \prod_{i=0}^{N/p-1} \big(a(\rho_i) - X \cdot b(\rho_i)\big) \pmod{X^2}. \end{aligned}$$

Since $a(X)$ and $b(X)$ are integer polynomials whose i-th coefficient is zero if p does not divide i, and that $F(X)$ (the p^r-th cyclotomic polynomial) has non-zero coefficients only for coefficients of X to the power of some multiple of p^{r-1}, we have that $a'(X) := a(X) \pmod{F(X)}$ and $b'(X) := b(X) \pmod{F(X)}$ will also be polynomials whose i-th coefficient is zero if p does not divide i.

So, if we define the polynomials V, U, such that $V(X^p) = a(X) \pmod{F(X)}$ and $U(X^p) = b(X) \pmod{f(X)}$, then we have reduced the original product of length N over v of degree $N-1$ down to a product of length N/p over the polynomials V and U, which have degree $N/p - 1$. This process can be applied recursively, until we end up with a final product of size $N/p^{r-1} = p-1$. This last product can then be computed in the naïve manner to obtain $g(X) \pmod{X^2}$. A similar reduction can also be applied to h.

The algorithm in Figure 1 shows how this reduction can be applied to compute g_0 and g_1. A simple modification to the algorithm will also allow h_1 to be computed at the same time. The proof of correctness for this is an obvious generalisation of the proof for the Gentry and Halevi reduction [6] and so is omitted.

4.3 m contains repeated factors

The algorithm described above can be used to speed up computation of g and h whenever m contains a repeated prime factor. If $m = p_1^{r_1} \cdots p_s^{r_s}$, then for every $r_i > 1$, $r_i - 1$ steps of the algorithm in Figure 1 can be carried out. So after each of these reductions the final product to be computed will be of size $(p_1 - 1) \cdots (p_s - 1)$. Clearly this speed improvement is most pronounced when $m = p^r$ for some small p, but it is nevertheless useful to note that gains can be made for any m with repeated prime factors.

5 Experiment Results

We now present some computational results for the relative performance of our new key generation method compared to the previous version. The original method was implemented in C++ using the MPFR library for arbitrary

COMPUTE-g-COEFFICIENTS(v, p, r)

$m \leftarrow p^r$
$F(X) \leftarrow \Phi_m(X)$
$U(X) \leftarrow 1$
$V(X) \leftarrow v(x)$
while $m > p$
    $v(X) \leftarrow V(X) \pmod{F(X)}$
    $V(X) \leftarrow \mathsf{resultant}_Y(v(Y),\, p \cdot X - p \cdot Y^p)/p^{p-1}$
    $q(X) \leftarrow U(X) \cdot V(X^p)/v(X)$
    for $i \leftarrow 0$ **to** $\deg(q)/p$
        $U_i \leftarrow q_{p \cdot i}$
    $U(X) \leftarrow U(X) \pmod{F(X)}$
    $U(X) \leftarrow U(X^{1/p})$
    $m \leftarrow m/p$
    $F(X) \leftarrow \Phi_m(X)$
// After the reduction, $p-1$ terms are left in the product.
$\rho \leftarrow e^{2 \cdot \pi \cdot \sqrt{-1}/p}$.
$g_0 \leftarrow \prod_{i=1}^{p-1} V(\rho^i), \qquad g_1 \leftarrow \sum_{i=1}^{p-1} U(\rho^i) \prod_{j \neq i} V(\rho^i)$
return g_0, g_1

Fig. 1. Algorithm to compute g_0 and g_1 when $m = p^r$.

precision floating point arithmetic, compiled using GCC 4.3.5. Our new method was coded with the computer algebra system Sage. Both algorithms were run on a high-powered server featuring an Intel Xeon E5620 processor running at 2.4GHz, with a 12MB cache and 48GB of memory.

We first describe the performance at four different values of m, each with different factorization properties. Namely, $m = 4391$, 5555, 6561 and 10125, which result in values of $n = \phi(m)$ in the range $[4000, 5400]$. The results (in minutes) for a value of $t = 400$ are given in Table 1.

m	4391	5555 $(= 5 \cdot 11 \cdot 101)$	6561 $(= 3^8)$	10125 $(= 3^4 \cdot 5^3)$
$\phi(m)$	4390	4000	4374	5400
Original Method	274	137	204	451
New Method	164	67	30	123
% Improvement	40%	51%	85%	72%

Table 1. Comparison of key generation methods for $t = 400$ and various values of m. Times are in minutes.

In Figure 2, we show how the performance of each algorithm as affected by t, for a fixed choice of m. We test each m with several different choices of the parameter t, the bit size of the generated coefficients. The bit length of a key will be approximately $t \cdot \phi(m)$, so increasing t increases the size of the numbers being computed on, and also requires a greater precision for any necessary floating point operations.

It is clear that our new method is significantly faster than the FFT method for all choices of m. In particular, when m contains many small repeated factors (here, for $m = 6561$ and 10125) the improvement gained is almost an order of magnitude. When the hybrid approach is taken, we see that the cost of recovering the key by inverting the matrix is far lower than that of using the second (inverse) FFT in the standard FFT method, and results in a speed increase of around 40-50%, as expected.

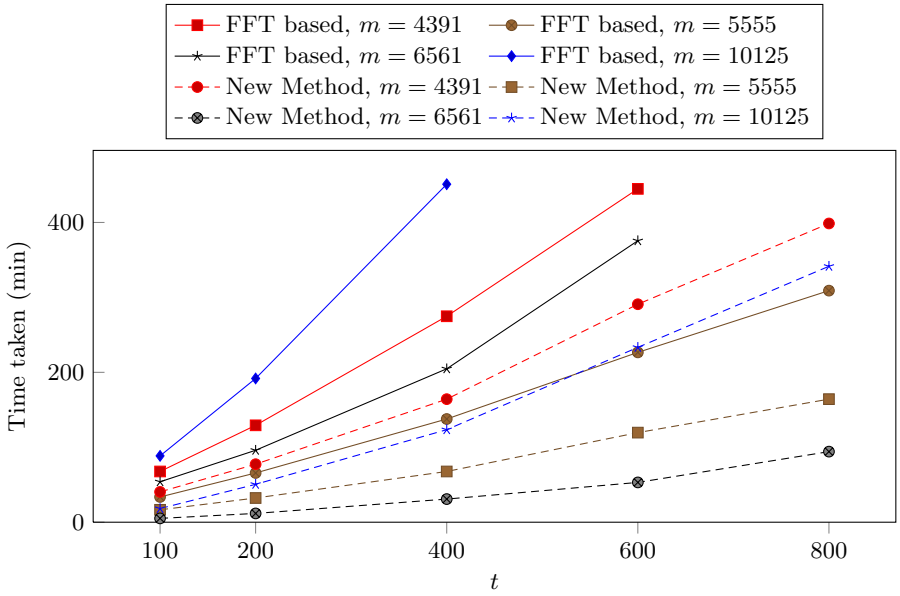

Fig. 2. Comparison of methods for various different values of m, as the parameter t increases.

6 Acknowledgements

The second author was supported by the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II and via an ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant COED–EP/I03126X, the Defense Advanced Research Projects Agency (DARPA) and

the Air Force Research Laboratory (AFRL) under agreement number FA8750-11-2-0079, and by a Royal Society Wolfson Merit Award. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, the U.S. Government, the European Commission or EPSRC.

References

1. H. Cohen. A Course in Computational Algebraic Number Theory. Springer GTM 138, 1993.
2. T.M. Apostol Introduction to Analytic Number Theory. Springer-Verlag, New York, 1976.
3. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. *Advances in Cryptology – Crypto 2011*, Springer LNCS 6841, 505–524, 2011.
4. C. Gentry. Fully homomorphic encryption using ideal lattices. *Symposium on Theory of Computing – STOC 2009*, ACM, 169–178, 2009.
5. C. Gentry. A fully homomorphic encryption scheme. PhD, Stanford University, 2009.
6. C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. *Advances in Cryptology – Eurocrypt 2011*, Springer LNCS 6632, 129–148, 2011.
7. I.J. Good. The interaction algorithm and practical Fourier analysis. *J.R. Stat. Soc.*, **20**, 361–372, 1958.
8. N. Ogura, G. Yamamoto, T. Kobayashi and S. Uchiyama. An improvement of key generation algorithm for Gentry's homomorphic encryption scheme. *Advances in Information and Computer Security – IWSEC 2010*, Springer LNCS 6434, 70–83, 2010.
9. C.M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proc. IEEE*, **56**, 1107–1108, 1968.
10. N.P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. *Public Key Cryptography – PKC 2010*, Springer LNCS 6056, 420–443, 2010
11. N.P. Smart and F. Vercauteren. Fully Homomorphic SIMD Operations. IACR e-print 2011/133.
12. L.H. Thomas. Using a computer to solve problems in physics. *Application of Digital Computers*, 1963.
13. W.F. Trench. An algorithm for the inversion of finite Toeplitz matrices. J. SIAM, **12**, 515-522, 1964.