Implementing Gentry's Fully-Homomorphic Encryption Scheme

Craig Gentry* and Shai Halevi*

IBM Research

Abstract. We describe a working implementation of a variant of Gentry's fully homomorphic encryption scheme (STOC 2009), similar to the variant used in an earlier implementation effort by Smart and Vercauteren (PKC 2010). Smart and Vercauteren implemented the underlying "somewhat homomorphic" scheme, but were not able to implement the bootstrapping functionality that is needed to get the complete scheme to work. We show a number of optimizations that allow us to implement all aspects of the scheme, including the bootstrapping functionality.

Our main optimization is a key-generation method for the underlying somewhat homomorphic encryption, that does not require full polynomial inversion. This reduces the asymptotic complexity from $\tilde{O}(n^{2.5})$ to $\tilde{O}(n^{1.5})$ when working with dimension-n lattices (and practically reducing the time from many hours/days to a few seconds/minutes). Other optimizations include a batching technique for encryption, a careful analysis of the degree of the decryption polynomial, and some space/time trade-offs for the fully-homomorphic scheme.

We tested our implementation with lattices of several dimensions, corresponding to several security levels. From a "toy" setting in dimension 512, to "small," "medium," and "large" settings in dimensions 2048, 8192, and 32768, respectively. The public-key size ranges in size from 70 Megabytes for the "small" setting to 2.3 Gigabytes for the "large" setting. The time to run one bootstrapping operation (on a 1-CPU 64-bit machine with large memory) ranges from 30 seconds for the "small" setting to 30 minutes for the "large" setting.

1 Introduction

Encryption schemes that support operations on encrypted data (aka homomorphic encryption) have a very wide range of applications in cryptography. This concept was introduced by Rivest et al. shortly after the discovery of public key cryptography [12], and many known public-key cryptosystems support either addition or multiplication of encrypted data. However, supporting both at the same time seems harder, and until very recently all the attempts at constructing so-called "*fully homomorphic*" encryption turned out to be insecure.

In 2009, Gentry described the first plausible construction of a fully homomorphic cryptosystem [3]. Gentry's construction consists of several steps: He first

* Supported by DARPA grant DARPA-BAA 10-81

constructed a "*somewhat homomorphic*" scheme that supports evaluating low-degree polynomials on the encrypted data, next he needed to "*squash*" the decryption procedure so that it can be expressed as a low-degree polynomial which is supported by the scheme, and finally he applied a "*bootstrapping*" transformation to obtain a fully homomorphic scheme. The crucial point in this process is to obtain a scheme that can evaluate polynomials of high-enough degree, and at the same time has decryption procedure that can be expressed as a polynomial of low-enough degree. Once the degree of polynomials that can be evaluated by the scheme exceeds the degree of the decryption polynomial (times two), the scheme is called "*bootstrappable*" and it can then be converted into a fully homomorphic scheme.

Toward a bootstrappable scheme, Gentry described in [3] a somewhat homomorphic scheme, which is roughly a GGH-type scheme [6, 8] over ideal lattices. Gentry later proved [4] that with an appropriate key-generation procedure, the security of that scheme can be (quantumly) reduced to the worst-case hardness of some lattice problems in ideal lattices.

This somewhat homomorphic scheme is not yet bootstrappable, so Gentry described in [3] a transformation to squash the decryption procedure, reducing the degree of the decryption polynomial. This is done by adding to the public key an additional hint about the secret key, in the form of a "*sparse subset-sum*" problem (SSSP). Namely the public key is augmented with a big set of vectors, such that there exists a very sparse subset of them that adds up to the secret key. A ciphertext of the underlying scheme can be "*post-processed*" using this additional hint, and the post-processed ciphertext can be decrypted with a low-degree polynomial, thus obtaining a bootstrappable scheme.

Stehlé and Steinfeld described in [14] two optimizations to Gentry's scheme, one that reduces the number of vectors in the SSSP instance, and another that can be used to reduce the degree of the decryption polynomial (at the expense of introducing a small probability of decryption errors). We mention that in our implementation we use the first optimization but not the second.[1] Some improvements to Gentry's key-generation procedure were discussed in [9].

1.1 The Smart-Vercauteren implementation

The first attempt to implement Gentry's scheme was made in 2010 by Smart and Vercauteren [13]. They chose to implement a variant of the scheme using "*principal-ideal lattices*" of prime determinant. Such lattices can be represented implicitly by just two integers (regardless of their dimension), and moreover Smart and Vercauteren described a decryption method where the secret key is represented by a single integer. Smart and Vercauteren were able to implement the underlying somewhat homomorphic scheme, but they were not able to support large enough parameters to make Gentry's squashing technique go

[1] The reason we do not use the second optimization is that the decryption error probability is too high for our parameter settings.

through. As a result they could not obtain a bootstrappable scheme or a fully homomorphic scheme.

One obstacle in the Smart-Vercauteren implementation was the complexity of key generation for the somewhat homomorphic scheme: For one thing, they must generate very many candidates before they find one whose determinant is prime. (One may need to try as many as $n^{1.5}$ candidates when working with lattices in dimension n.) And even after finding one, the complexity of computing the secret key that corresponds to this lattice is at least $\tilde{\Theta}(n^{2.5})$ for lattices in dimension n. For both of these reasons, they were not able to generate keys in dimensions $n > 2048$.

Moreover, Smart and Vercauteren estimated that the squashed decryption polynomial will have degree of a few hundreds, and that to support this procedure with their parameters they need to use lattices of dimension at least $n = 2^{27} (\approx 1.3 \times 10^8)$, which is well beyond the capabilities of the key-generation procedure.

1.2 Our implementation

We continue in the same direction of the Smart-Vercauteren implementation and describe optimizations that allow us to implement also the squashing part, thereby obtaining a bootstrappable scheme and a fully homomorphic scheme.

For key-generation, we present a new faster algorithm for computing the secret key, and also eliminate the requirement that the determinant of the lattice be prime. We also present many simplifications and optimizations for the squashed decryption procedure, and as a result our decryption polynomial has degree only fifteen. Finally, our choice of parameters is somewhat more aggressive than Smart and Vercauteren (which we complement by analyzing the complexity of known attacks).

Differently from [13], we decouple the dimension n from the size of the integers that we choose during key generation. Decoupling these two parameters lets us decouple functionality from security. Namely, we can obtain bootstrappable schemes in any given dimension, but of course the schemes in low dimensions will not be secure. Our (rather crude) analysis suggests that the scheme may be practically secure at dimension $n = 2^{13}$ or $n = 2^{15}$, and we put this analysis to the test by publishing a few challenges in dimensions from 512 up to 2^{15}.

1.3 Organization

We give some background in Section 2, and then describe our implementation of the underlying "somewhat homomorphic" encryption scheme in Sections 3 through 7. A description of our optimizations that are specific to the bootstrapping functionality appears in the full version of this report [5].

2 Background

Notations. Throughout this report we use '$\cdot$' to denote scalar multiplication and '$\times$' to denote any other type of multiplication. For integers z, d, we denote the

reduction of z modulo d by either $[z]_d$ or $\langle z \rangle_d$. We use $[z]_d$ when the operation maps integers to the interval $[-d/2,\ d/2)$, and use $\langle z \rangle_d$ when the operation maps integers to the interval $[0, d)$. We use the generic "z mod d" when the specific interval does not matter (e.g., mod 2). For example we have $[13]_5 = -2$ vs. $\langle 13 \rangle_5 = 3$, but $[9]_7 = \langle 9 \rangle_7 = 2$.

For a rational number q, we denote by $\lceil q \rfloor$ the rounding of q to the nearest integer, and by $[q]$ we denote the distance between q and the nearest integer. That is, if $q = \frac{a}{b}$ then $[q] \stackrel{\text{def}}{=} \frac{[a]_b}{b}$ and $\lceil q \rfloor \stackrel{\text{def}}{=} q - [q]$. For example, $\lceil \frac{13}{5} \rfloor = 3$ and $[\frac{13}{5}] = \frac{-2}{5}$. These notations are extended to vectors in the natural way: for example if $\boldsymbol{q} = \langle q_0, q_1, \ldots, q_{n-1} \rangle$ is a rational vector then rounding is done coordinate-wise, $\lceil \boldsymbol{q} \rfloor = \langle \lceil q_0 \rfloor, \lceil q_1 \rfloor, \ldots, \lceil q_{n-1} \rfloor \rangle$.

2.1 Lattices

A full-rank n-dimensional *lattice* is a discrete subgroup of $\mathbb{R}^n$, concretely represented as the set of all integer linear combinations of some *basis* $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^n$ of linearly independent vectors. Viewing the vectors $\boldsymbol{b}_i$ as the rows of a matrix $B \in \mathbb{R}^{n \times n}$, we have: $L = \mathcal{L}(B) = \{\boldsymbol{y} \times B \ :\ \boldsymbol{y} \in \mathbb{Z}^n\}$.

Every lattice (of dimension $n > 1$) has an infinite number of lattice bases. If B_1 and B_2 are two lattice bases of L, then there is some unimodular matrix U (i.e., U has integer entries and $\det(U) = \pm 1$) satisfying $B_1 = U \times B_2$. Since U is unimodular, $|\det(B_i)|$ is invariant for different bases of L, and we may refer to it as $\det(L)$. This value is precisely the size of the quotient group $\mathbb{Z}^n/L$ if L is an integer lattice. To basis B of lattice L we associate the half-open parallelepiped $\mathcal{P}(B) \leftarrow \{\sum_{i=1}^n x_i \boldsymbol{b}_i : x_i \in [-1/2, 1/2)\}$. The volume of $\mathcal{P}(B)$ is precisely $\det(L)$.

For $\boldsymbol{c} \in \mathbb{R}^n$ and basis B of L, we use $\boldsymbol{c} \bmod B$ to denote the unique vector $\boldsymbol{c'} \in \mathcal{P}(B)$ such that $\boldsymbol{c} - \boldsymbol{c'} \in L$. Given $\boldsymbol{c}$ and B, $\boldsymbol{c} \bmod B$ can be computed efficiently as $\boldsymbol{c} - \lfloor \boldsymbol{c} \times B^{-1} \rceil \times B = [\boldsymbol{c} \times B^{-1}] \times B$. (Recall that $\lfloor \cdot \rceil$ means rounding to the nearest integer and $[\cdot]$ is the fractional part.)

Every full-rank lattice has a unique Hermite normal form (HNF) basis where $b_{i,j} = 0$ for all $i < j$ (lower-triangular), $b_{j,j} > 0$ for all j, and for all $i > j$ $b_{i,j} \in [-b_{j,j}/2, +b_{j,j}/2)$. Given any basis B of L, one can compute $\mathrm{HNF}(L)$ efficiently via Gaussian elimination. The HNF is in some sense the "least revealing" basis of L, and thus typically serves as the public key representation of the lattice [8].

Short vectors and Bounded Distance Decoding. The length of the shortest nonzero vector in a lattice L is denoted $\lambda_1(L)$, and Minkowski's theorem says that for any n-dimensional lattice L ($n > 1$) we have $\lambda_1(L) < \sqrt{n} \cdot \det(L)^{1/n}$. Heuristically, for random lattices the quantity $\det(L)^{1/n}$ serves as a threshold: for $t \ll \det(L)^{1/n}$ we don't expect to find any nonzero vectors in L of size t, but for $t \gg \det(L)^{1/n}$ we expect to find exponentially many vectors in L of size t.

In the "*bounded distance decoding*" problem (BDDP), one is given a basis B of some lattice L, and a vector $\boldsymbol{c}$ that is very close to some lattice point of L, and the goal is to find the point in L nearest to $\boldsymbol{c}$. In the promise problem γ-BDDP, we have a parameter $\gamma > 1$ and the promise that $\mathrm{dist}(L, \boldsymbol{c}) \stackrel{\text{def}}{=} \min_{\boldsymbol{v} \in L}\{\|\boldsymbol{c} - \boldsymbol{v}\|\} \leq$

$\det(L)^{1/n}/\gamma$. (BDDP is often defined with respect to λ_1 rather than with respect to $\det(L)^{1/n}$, but the current definition is more convenient in our case.)

Gama and Nguyen conducted extensive experiments with lattices in dimensions 100-400 [2], and concluded that for those dimensions it is feasible to solve γ-BDDP when $\gamma > 1.01^n \approx 2^{n/70}$. More generally, the best algorithms for solving the γ-BDDP in n-dimensional lattices take time exponential in $n/\log\gamma$. Specifically, currently known algorithms can solve dimension-n γ-BDDP in time 2^k up to $\gamma = 2^{\frac{\mu n}{k/\log k}}$, where μ is a parameter that depends on the exact details of the algorithm. (Extrapolating from the Gama-Nguyen experiments, we expect something like $\mu \in [0.1, 0.2]$.)

2.2 Ideal lattices

Let $f(x)$ be an integer monic irreducible polynomial of degree n. In this paper, we use $f(x) = x^n + 1$, where n is a power of 2. Let R be the ring of integer polynomials modulo $f(x)$, $R \stackrel{\text{def}}{=} \mathbb{Z}[x]/(f(x))$. Each element of R is a polynomial of degree at most $n-1$, and thus is associated to a coefficient vector in $\mathbb{Z}^n$. This way, we can view each element of R as being both a polynomial and a vector. For $\boldsymbol{v}(x)$, we let $\|\boldsymbol{v}\|$ be the Euclidean norm of its coefficient vector. For every ring R, there is an associated expansion factor $\gamma_{\mathsf{Mult}}(R)$ such that $\|\boldsymbol{u} \times \boldsymbol{v}\| \leq \gamma_{\mathsf{Mult}}(R) \cdot \|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|$, where $\times$ denotes multiplication in the ring. When $f(x) = x^n + 1$, $\gamma_{\mathsf{Mult}}(R)$ is $\sqrt{n}$. However, for "random vectors" $\boldsymbol{u}, \boldsymbol{v}$ the expansion factor is typically much smaller, and our experiments suggest that we typically have $\|\boldsymbol{u} \times \boldsymbol{v}\| \approx \|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|$.

Let I be an ideal of R – that is, a subset of R that is closed under addition and multiplication by elements of R. Since I is additively closed, the coefficient vectors associated to elements of I form a *lattice*. We call I an *ideal lattice* to emphasize this object's dual nature as an algebraic ideal and a lattice.[2] Ideals have additive structure as lattices, but they also have multiplicative structure. The *product* IJ of two ideals I and J is the additive closure of the set $\{\boldsymbol{v} \times \boldsymbol{w} : \boldsymbol{v} \in I, \boldsymbol{w} \in J\}$, where '$\times$' is ring multiplication. To simplify things, we will use *principal* ideals of R – i.e., ideals with a single generator. The ideal $(\boldsymbol{v})$ generated by $\boldsymbol{v} \in R$ corresponds to the lattice generated by the vectors $\{\boldsymbol{v}_i \stackrel{\text{def}}{=} \boldsymbol{v} \times x^i \bmod f(x) : i \in [0, n-1]\}$; we call this the *rotation basis* of the ideal lattice $(\boldsymbol{v})$.

Let K be a field containing the ring R (in our case $K = \mathbb{Q}[x]/(f(x))$). The *inverse* of an ideal $I \subseteq R$ is $I^{-1} = \{\boldsymbol{w} \in K : \forall \boldsymbol{v} \in I, \boldsymbol{v} \times \boldsymbol{w} \in R\}$. The inverse of a principal ideal $(\boldsymbol{v})$ is given by $(\boldsymbol{v}^{-1})$, where the inverse $\boldsymbol{v}^{-1}$ is taken in the field K.

2.3 GGH-type cryptosystems

We briefly recall Micciancio's "cleaned-up version" of GGH cryptosystems [6, 8]. The secret and public keys are "good" and "bad" bases of some lattice L.

[2] Alternative representations of an ideal lattice are possible – e.g., see [11, 7].

More specifically, the key-holder generates a good basis by choosing B_{sk} to be a basis of short, "nearly orthogonal" vectors. Then it sets the public key to be the Hermite normal form of the same lattice, $B_{pk} \stackrel{\text{def}}{=} \text{HNF}(\mathcal{L}(B_{sk}))$.

A ciphertext in a GGH-type cryptosystem is a vector $\boldsymbol{c}$ close to the lattice $\mathcal{L}(B_{pk})$, and the message which is encrypted in this ciphertext is somehow embedded in the distance from $\boldsymbol{c}$ to the nearest lattice vector. To encrypt a message m, the sender chooses a short "error vector" $\boldsymbol{e}$ that encodes m, and then computes the ciphertext as $\boldsymbol{c} \leftarrow \boldsymbol{e} \bmod B_{pk}$. Note that if $\boldsymbol{e}$ is short enough (i.e., less than $\lambda_1(L)/2$), then it is indeed the distance between $\boldsymbol{c}$ and the nearest lattice point.

To decrypt, the key-holder uses its "good" basis B_{sk} to recover $\boldsymbol{e}$ by setting $\boldsymbol{e} \leftarrow \boldsymbol{c} \bmod B_{sk}$, and then recovers m from $\boldsymbol{e}$. The reason decryption works is that, if the parameters are chosen correctly, then the parallelepiped $\mathcal{P}(B_{sk})$ of the secret key will be a "plump" parallelepiped that contains a sphere of radius bigger than $\|\boldsymbol{e}\|$, so that $\boldsymbol{e}$ is the point inside $\mathcal{P}(B_{sk})$ that equals $\boldsymbol{c}$ modulo L. On the other hand, the parallelepiped $\mathcal{P}(B_{pk})$ of the public key will be very skewed, and will not contain a sphere of large radius, making it useless for solving BDDP.

2.4 Gentry's somewhat-homomorphic cryptosystem

Gentry's somewhat homomorphic encryption scheme [3] can be seen as a GGH-type scheme over ideal lattices. The public key consists of a "bad" basis B_{pk} of an ideal lattice J, along with some basis B_I of a "small" ideal I (which is used to embed messages into the error vectors). For example, the small ideal I can be taken to be $I = (2)$, the set of vectors with all even coefficients.

A ciphertext in Gentry's scheme is a vector close to a J-point, with the message being embedded in the distance to the nearest lattice point. More specifically, the plaintext space is $\{0,1\}$, which is embedded in $R/I = \{0,1\}^n$ by encoding 0 as 0^n and 1 as $0^{n-1}1$. For an encoded bit $\boldsymbol{m} \in \{0,1\}^n$ we set $\boldsymbol{e} = 2\boldsymbol{r} + \boldsymbol{m}$ for a random small vector $\boldsymbol{r}$, and then output the ciphertext $\boldsymbol{c} \leftarrow \boldsymbol{e} \bmod B_{pk}$.

The secret key in Gentry's scheme (that plays the role of the "good basis" of J) is just a short vector $\boldsymbol{w} \in J^{-1}$. Decryption involves computing the fractional part $[\boldsymbol{w} \times \boldsymbol{c}]$. Since $\boldsymbol{c} = \boldsymbol{j} + \boldsymbol{e}$ for some $j \in J$, then $\boldsymbol{w} \times \boldsymbol{c} = \boldsymbol{w} \times \boldsymbol{j} + \boldsymbol{w} \times \boldsymbol{e}$. But $\boldsymbol{w} \times \boldsymbol{j}$ is in R and thus an integer vector, so $\boldsymbol{w} \times \boldsymbol{c}$ and $\boldsymbol{w} \times \boldsymbol{e}$ have the same fractional part, $[\boldsymbol{w} \times \boldsymbol{c}] = [\boldsymbol{w} \times \boldsymbol{e}]$. If $\boldsymbol{w}$ and $\boldsymbol{e}$ are short enough – in particular, if we have the guarantee that all of the coefficients of $\boldsymbol{w} \times \boldsymbol{e}$ have magnitude less than $1/2$ – then $[\boldsymbol{w} \times \boldsymbol{e}]$ equals $\boldsymbol{w} \times \boldsymbol{e}$ exactly. From $\boldsymbol{w} \times \boldsymbol{e}$, the decryptor can multiply by $\boldsymbol{w}^{-1}$ to recover $\boldsymbol{e}$, and then recover $\boldsymbol{m} \leftarrow \boldsymbol{e} \bmod 2$. The actual decryption procedure from [3] is slightly different, however. Specifically, $\boldsymbol{w}$ is "tweaked" so that decryption can be implemented as $\boldsymbol{m} \leftarrow \boldsymbol{c} - [\boldsymbol{w} \times \boldsymbol{c}] \bmod 2$ (when $I = (2)$).

The reason that this scheme is somewhat homomorphic is that for two ciphertexts $\boldsymbol{c}_1 = \boldsymbol{j}_1 + \boldsymbol{e}_1$ and $\boldsymbol{c}_2 = \boldsymbol{j}_2 + \boldsymbol{e}_2$, their sum is $\boldsymbol{j}_3 + \boldsymbol{e}_3$ where $\boldsymbol{j}_3 = \boldsymbol{j}_1 + \boldsymbol{j}_2 \in J$ and $\boldsymbol{e}_3 = \boldsymbol{e}_1 + \boldsymbol{e}_2$ is small. Similarly, their product is $\boldsymbol{j}_4 + \boldsymbol{e}_4$ where $\boldsymbol{j}_4 = \boldsymbol{j}_1 \times (\boldsymbol{j}_2 + \boldsymbol{e}_2) + \boldsymbol{e}_1 \times \boldsymbol{j}_2 \in J$ and $\boldsymbol{e}_4 = \boldsymbol{e}_1 \times \boldsymbol{e}_2$ is still small. If fresh encrypted

ciphertexts are very very close to the lattice, then it is possible to add and multiply ciphertexts for a while before the error grows beyond the decryption radius of the secret key.

The Smart-Vercauteren Variant Smart and Vercauteren [13] work over the ring $R = \mathbb{Z}[x]/f_n(x)$, where $f_n(x) = x^n + 1$ and n is a power of two. The ideal J is set as a principal ideal by choosing a vector $\boldsymbol{v}$ at random from some n-dimensional cube, subject to the condition that the determinant of $(\boldsymbol{v})$ is prime, and then setting $J = (\boldsymbol{v})$. It is known that such ideals can be implicitly represented by only two integers, namely the determinant $d = \det(J)$ and a root r of $f_n(x)$ modulo d. (An easy proof of this fact "from first principles" can be derived from our Lemma 1 below.) Specifically, the Hermite normal form of this ideal lattice is

$$\mathrm{HNF}(J) = \begin{bmatrix} d & 0 & 0 & 0 & \cdots & 0 \\ -r & 1 & 0 & 0 & \cdots & 0 \\ -[r^2]_d & 0 & 1 & 0 & \cdots & 0 \\ -[r^3]_d & 0 & 0 & 1 & \cdots & 0 \\ & & & & \ddots & \\ -[r^{n-1}]_d & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{1}$$

It is easy to see that reducing a vector $\boldsymbol{a}$ modulo $\mathrm{HNF}(J)$ consists of evaluating the associated polynomial $a(x)$ at the point r modulo d, then outputting the vector $\langle [a(r)]_d, 0, 0, \ldots, 0\rangle$ (see Section 5). Hence encryption of a vector $\langle m, 0, 0, \ldots, 0\rangle$ with $m \in \{0, 1\}$ can be done by choosing a random small polynomial $u(x)$ and evaluating it at r, then outputting the integer $c \leftarrow [2u(r) + m]_d$.

Smart and Vercauteren also describe a decryption procedure that uses a single integer w as the secret key, setting $m \leftarrow (c - \lceil cw/d \rfloor) \bmod 2$. Jumping ahead, we note that our decryption procedure from Section 6 is very similar, except that for convenience we replace the rational division cw/d by modular multiplication $[cw]_d$.

3 Key generation

We adopt the Smart-Vercauteren approach [13], in that we also use principal-ideal lattices in the ring of polynomials modulo $f_n(x) \stackrel{\text{def}}{=} x^n+1$ with n a power of two. We do not require that these principal-ideal lattices have prime determinant, instead we only need the Hermite normal form to have the same form as in Equation (1). During key-generation we choose $\boldsymbol{v}$ at random in some cube, verify that the HNF has the right form, and work with the principal ideal $(\boldsymbol{v})$. We have two parameters: the dimension n, which must be a power of two, and the bit-size t of coefficients in the generating polynomial. Key-generation consists of the following steps:

1. Choose a random n-dimensional integer lattice $\boldsymbol{v}$, where each entry v_i is chosen at random as a t-bit (signed) integer. With this vector $\boldsymbol{v}$ we associate

the formal polynomial $v(x) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} v_i x^i$, as well as the rotation basis:

$$V = \begin{bmatrix} v_0 & v_1 & v_2 & \cdots & v_{n-1} \\ -v_{n-1} & v_0 & v_1 & \cdots & v_{n-2} \\ -v_{n-2} & -v_{n-1} & v_0 & \cdots & v_{n-3} \\ & & & \ddots & \\ -v_1 & -v_2 & -v_3 & & v_0 \end{bmatrix} \tag{2}$$

The i'th row is a cyclic shift of $\boldsymbol{v}$ by i positions to the right, with the "overflow entries" negated. Note that the i'th row corresponds to the coefficients of the polynomial $v_i(x) = v(x) \times x^i \pmod{f_n(x)}$. Note that just like V itself, the entire lattice $\mathcal{L}(V)$ is also closed under "rotation": Namely, for any vector $\langle u_0, u_1, \ldots, u_{n-1} \rangle \in \mathcal{L}(V)$, also the vector $\langle -u_{n-1}, u_0, \ldots, u_{n-2} \rangle$ is in $\mathcal{L}(V)$.

2. Next we compute the scaled inverse of $v(x)$ modulo $f_n(x)$, namely an integer polynomial $w(x)$ of degree at most $n-1$, such that $w(x) \times v(x) = \mathsf{constant} \pmod{f_n(x)}$. Specifically, this constant is the determinant of the lattice $\mathcal{L}(V)$, which must be equal to the resultant of the polynomials $v(x)$ and $f_n(x)$ (since f_n is monic). Below we denote the resultant by d, and denote the coefficient-vector of $w(x)$ by $\boldsymbol{w} = \langle w_0, w_1, \ldots, w_{n-1} \rangle$. It is easy to check that the matrix

$$W = \begin{bmatrix} w_0 & w_1 & w_2 & \cdots & w_{n-1} \\ -w_{n-1} & w_0 & w_1 & \cdots & w_{n-2} \\ -w_{n-2} & -w_{n-1} & w_0 & \cdots & w_{n-3} \\ & & & \ddots & \\ -w_1 & -w_2 & -w_3 & & w_0 \end{bmatrix} \tag{3}$$

is the scaled inverse of V, namely $W \times V = V \times W = d \cdot I$. One way to compute the polynomial $w(x)$ is by applying the extended Euclidean-GCD algorithm (for polynomials) to $v(x)$ and $f_n(x)$. See Section 4 for a more efficient method of computing $w(x)$.

3. We also check that this is a good generating polynomial. We consider $\boldsymbol{v}$ to be good if the Hermite-Normal-form of V has the same form as in Equation (1), namely all except the leftmost column equal to the identity matrix. See below for a simple check that the $\boldsymbol{v}$ is good, in our implementation we test this condition while computing the inverse.
   It was observed by Nigel Smart that the HNF has the correct form whenever the determinant is odd and square-free. Indeed, in our tests this condition was met with probability roughly 0.5, irrespective of the dimension and bit length, with the failure cases usually due to the determinant of V being even.

Checking the HNF. In Lemma 1 below we prove that the HNF of the lattice $\mathcal{L}(V)$ has the right form if and only if the lattice contains a vector of the form $\langle -r, 1, 0, \ldots, 0 \rangle$. Namely, if and only if there exists an integer vector $\boldsymbol{y}$ and another integer r such that

$$\boldsymbol{y} \times V = \langle -r, 1, 0, \ldots, 0 \rangle$$

Multiplying the last equation on the right by W, we get the equivalent condition

$$\begin{aligned} & \boldsymbol{y} \times V \times W = \langle -r, 1, 0 \ldots, 0 \rangle \times W \qquad (4) \\ \Leftrightarrow \quad & \boldsymbol{y} \times (dI) \;=\; d \cdot \boldsymbol{y} = -r \cdot \langle w_0, w_1, w_2, \ldots, w_{n-1} \rangle + \langle -w_{n-1}, w_0, w_1, \ldots, w_{n-2} \rangle \end{aligned}$$

In other words, there must exists an integer r such that the second row of W minus r times the first row yields a vector of integers that are all divisible by d:

$$\begin{aligned} & -r \cdot \langle w_0, w_1, w_2, \ldots, w_{n-1} \rangle + \langle -w_{n-1}, w_0, w_1, \ldots, w_{n-2} \rangle \;=\; 0 \pmod{d} \\ \Leftrightarrow\; & -r \cdot \langle w_0, w_1, w_2, \ldots, w_{n-1} \rangle \;=\; \langle w_{n-1}, -w_0, -w_1, \ldots, -w_{n-2} \rangle \pmod{d} \end{aligned}$$

The last condition can be checked easily: We compute $r := w_0/w_1 \bmod d$ (assuming that w_1 has an inverse modulo d), then check that $r \cdot w_{i+1} = w_i \pmod{d}$ holds for all $i = 1, \ldots, n-2$ and also $-r \cdot w_0 = w_{n-1} \pmod{d}$. Note that this means in particular that $r^n = -1 \pmod{d}$. (In our implementation we actually test only that last condition, instead of testing all the equalities $r \cdot w_{i+1} = w_i \pmod{d}$.)

Lemma 1. *The Hermite normal form of the matrix V from Equation (2) is equal to the identity matrix in all but the leftmost column, if and only if the lattice spanned by the rows of V contains a vector of the form $\boldsymbol{r} = \langle -r, 1, 0 \ldots, 0 \rangle$.*

Proof. Let B be the Hermite normal form of V. Namely, B is lower triangular matrix with non-negative diagonal entries, where the rows of B span the same lattice as the rows of V, and the absolute value of every entry under the diagonal in B is no more than half the diagonal entry above it. This matrix B can be obtained from V by a sequence of elementary row operations, and it is unique. It is easy to see that the existence of a vector $\boldsymbol{r}$ of this form is necessary: indeed the second row of B must be of this form (since B is equal the identity in all except the leftmost column). We now prove that this condition is also sufficient.

It is clear that the vector $d \cdot \boldsymbol{e_1} = \langle d, 0, \ldots, 0 \rangle$ belongs to $\mathcal{L}(V)$: in particular we know that $\langle w_0, w_1, \ldots, w_{n-1} \rangle \times V = \langle d, 0, \ldots, 0 \rangle$. Also, by assumption we have $\boldsymbol{r} = -r \cdot \boldsymbol{e_1} + \boldsymbol{e_2} \in \mathcal{L}(V)$, for some integer r. Note that we can assume without loss of generality that $-d/2 \le r < d/2$, since otherwise we could subtract from $\boldsymbol{r}$ multiples of the vector $d \cdot \boldsymbol{e_1}$ until this condition is satisfied:

$$\begin{aligned} & \langle -r \quad 1\ 0\ \ldots\ 0 \rangle \\ -\kappa \cdot & \langle \quad d \quad 0\ 0\ \ldots\ 0 \rangle \\ = \; & \overline{\langle [-r]_d\ 1\ 0\ \ldots\ 0 \rangle} \end{aligned}$$

For $i = 1, 2, \ldots, n-1$, denote $r_i \stackrel{\text{def}}{=} [r^i]_d$. Below we will prove by induction that for all $i = 1, 2, \ldots, n-1$, the lattice $\mathcal{L}(V)$ contains the vector:

$$\boldsymbol{r_i} \stackrel{\text{def}}{=} -r_i \cdot \boldsymbol{e_1} + \boldsymbol{e}_{i+1} \;=\; \underbrace{\langle -r_i, 0 \ldots 0, 1, 0 \ldots 0 \rangle}_{1 \text{ in the } i{+}1'\text{st position}}.$$

Placing all these vectors $\boldsymbol{r_i}$ at the rows of a matrix, we got exactly the matrix B that we need:

$$B = \begin{bmatrix} d & 0 & 0 & 0 \\ -r_1 & 1 & 0 & 0 \\ -r_2 & 0 & 1 & 0 \\ & & \ddots & \\ -r_{n-1} & 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

B is equal to the identity except in the leftmost column, its rows are all vectors in $\mathcal{L}(V)$ (so they span a sub-lattice), and since B has the same determinant as V then it cannot span a proper sub-lattice, it must therefore span $\mathcal{L}(V)$ itself.

It is left to prove the inductive claim. For $i = 1$ we set $\boldsymbol{r_1} \stackrel{\text{def}}{=} \boldsymbol{r}$ and the claim follow from our assumption that $\boldsymbol{r} \in \mathcal{L}(V)$. Assume now that it holds for some $i \in [1, n-2]$ and we prove for $i+1$. Recall that the lattice $\mathcal{L}(V)$ is closed under rotation, and since $\boldsymbol{r_i} = -r_i\boldsymbol{e_1} + \boldsymbol{e}_{i+1} \in \mathcal{L}(V)$ then the right-shifted vector $\boldsymbol{s}_{i+1} \stackrel{\text{def}}{=} -r_i\boldsymbol{e_2} + \boldsymbol{e}_{i+2}$ is also in $\mathcal{L}(V)$.[3] Hence $\mathcal{L}(V)$ contains also the vector

$$\boldsymbol{s}_{i+1} + r_i \cdot \boldsymbol{r} \;=\; (-r_i\boldsymbol{e_2} + \boldsymbol{e_{i+2}}) + r_i(-r\boldsymbol{e_1} + \boldsymbol{e_2}) = \;= -r_i r \cdot \boldsymbol{e_1} + \boldsymbol{e}_{i+2}$$

We can now reduce the first entry in this vector modulo d, by adding/subtracting the appropriate multiple of $d \cdot \boldsymbol{e_1}$ (while still keeping it in the lattice), thus getting the lattice vector

$$[-r \cdot r_i]_d \cdot \boldsymbol{e_1} + \boldsymbol{e_{i+2}} \;=\; -[r^{i+1}]_d \cdot \boldsymbol{e_1} + \boldsymbol{e_{i+2}} \;=\; \boldsymbol{r}_{i+1} \;\in\; \mathcal{L}(V)$$

This concludes the proof.

Remark 1. Note that the proof of Lemma 1 shows in particular that if the Hermite normal form of V is equal to the identity matrix in all but the leftmost column, then it must be of the form specified in Equation (5). Namely, the first column is $\langle d, -r_1, -r_2, \ldots, -r_{n-1}\rangle^t$, with $r_i = [r^i]_d$ for all i. Hence this matrix can be represented implicitly by the two integers d and r.

3.1 The public and secret keys

In principle the public key is the Hermite normal form of V, but as we explain in Remark 1 and Section 5 it is enough to store for the public key only the two integers d, r. Similarly, in principle the secret key is the pair $(\boldsymbol{v}, \boldsymbol{w})$, but as we explain in Section 6.1 it is sufficient to store only a single (odd) coefficient of $\boldsymbol{w}$ and discard $\boldsymbol{v}$ altogether.

4 Inverting the polynomial $v(x)$

The fastest known methods for inverting the polynomial $v(x)$ modulo $f_n(x) = x^n + 1$ are based on FFT: We can evaluate $v(x)$ at all the roots of $f_n(x)$ (either

[3] This is a circular shift, since $i \leq n-2$ and hence the rightmost entry in $\boldsymbol{r_i}$ is zero.

over the complex field or over some finite field), then compute $w^*(\rho) = 1/v(\rho)$ (where inversion is done over the corresponding field), and then interpolate $w^* = v^{-1}$ from all these values. If the resultant of v and f_n has N bits, then this procedure will take $O(n \log n)$ operations over $O(N)$-bit numbers, for a total running time of $\tilde{O}(nN)$. This is close to optimal in general, since just writing out the coefficients of the polynomial w^* takes time $O(nN)$. However, in Section 6.1 we show that it is enough to use for the secret key only one of the coefficients of $w = d \cdot w^*$ (where $d = \mathsf{resultant}(v, f_n)$). This raises the possibility that we can compute this one coefficient in time quasi-linear in N rather than quasi-linear in nN. Although polynomial inversion is very well researched, as far as we know this question of computing just one coefficient of the inverse was not tackled before. Below we describe an algorithm for doing just that.

The approach for the procedure below is to begin with the polynomial v that has n small coefficients, and proceed in steps where in each step we halve the number of coefficients to offset the fact that the bit-length of the coefficients approximately doubles. Our method relies heavily on the special form of $f_n(x) = x^n + 1$, with n a power of two. Let $\rho_0, \rho_1, \ldots, \rho_{n-1}$ be roots of $f_n(x)$ over the complex field: That is, if ρ is some primitive $2n$'th root of unity then $\rho_i = \rho^{2i+1}$. Note that the roots r_i satisfy that $\rho_{i+\frac{n}{2}} = -\rho_i$ for all i, and more generally for every index i (with index arithmetic modulo n) and every $j = 0, 1, \ldots, \log n$, if we denote $n_j \stackrel{\text{def}}{=} n/2^j$ then it holds that

$$\left(\rho_{i+\,n_j/2}\right)^{2^j} \;=\; \left(\rho^{2i+n_j+1}\right)^{2^j} \;=\; \left(\rho^{2i+1}\right)^{2^j} \cdot \rho^n \;=\; -\left(\rho_i^{\,2^j}\right) \tag{6}$$

The method below takes advantage of Equation (6), as well as a connection between the coefficients of the scaled inverse w and those of the formal polynomial

$$g(z) \;\stackrel{\text{def}}{=}\; \prod_{i=0}^{n-1} \big(v(\rho_i) - z\big).$$

We invert $v(x)$ mod $f_n(x)$ by computing the lower two coefficients of $g(z)$, then using them to recover both the resultant and (one coefficient of) the polynomial $w(x)$, as described next.

Step one: the polynomial $g(z)$. Note that although the polynomial $g(z)$ is defined via the complex numbers ρ_i, the coefficients of $g(z)$ are all integers. We begin by showing how to compute the lower two coefficients of $g(z)$, namely the polynomial $g(z)$ mod z^2. We observe that since $\rho_{i+\frac{n}{2}} = -\rho_i$ then we can write $g(z)$ as

$$\begin{aligned} g(z) &= \prod_{i=0}^{\frac{n}{2}-1} (v(\rho_i) - z)(v(-\rho_i) - z) \\ &= \prod_{i=0}^{\frac{n}{2}-1} \Big(\underbrace{v(\rho_i)v(-\rho_i)}_{a(\rho_i)} - z\underbrace{(v(\rho_i) + v(-\rho_i))}_{b(\rho_i)} + z^2 \Big) \;=\; \prod_{i=0}^{\frac{n}{2}-1} \Big(a(\rho_i) - z b(\rho_i) \Big) \pmod{z^2} \end{aligned}$$

We observe further that for both the polynomials $a(x) = v(x)v(-x)$ and $b(x) = v(x) + v(-x)$, all the odd powers of x have zero coefficients. Moreover, the same equalities as above hold if we use $A(x) = a(x) \bmod f_n(x)$ and $B(x) = b(x) \bmod f_n(x)$ instead of $a(x)$ and $b(x)$ themselves (since we only evaluate these polynomials in roots of f_n), and also for A, B all the odd powers of x have zero coefficients (since we reduce modulo $f_n(x) = x^n + 1$ with n even).

Thus we can consider the polynomials $\hat{v}, \tilde{v}$ that have half the degree and only use the nonzero coefficients of A, B, respectively. Namely they are defined via $\hat{v}(x^2) = A(x)$ and $\tilde{v}(x^2) = B(x)$. Thus we have reduced the task of computing the n-product involving the degree-n polynomial $v(x)$ to computing a product of only $n/2$ terms involving the degree-$n/2$ polynomials $\hat{v}(x), \tilde{v}(x)$. Repeating this process recursively, we obtain the polynomial $g(z) \bmod z^2$. The details of this process are described in Section 4.1 below.

Step two: recovering d and w_0. Recall that if $v(x)$ is square free then $d = \mathsf{resultant}(v, f_n) = \prod_{i=0}^{n-1} v(\rho_i)$, which is exactly the free term of $g(z)$, $g_0 = \prod_{i=0}^{n-1} v(\rho_i)$.

Recall also that the linear term in $g(z)$ has coefficient $g_1 = \sum_{i=0}^{n-1} \prod_{j \neq i} v(\rho_i)$. We next show that the free term of $w(x)$ is $w_0 = g_1/n$. First, we observe that g_1 equals the sum of w evaluated in all the roots of f_n, namely

$$g_1 = \sum_{i=0}^{n-1} \prod_{j \neq i} v(\rho_j) = \sum_{i=0}^{n-1} \frac{\prod_{j=0}^{n-1} v(\rho_j)}{v(\rho_i)} \overset{\text{(a)}}{=} \sum_{i=0}^{n-1} \frac{d}{v(\rho_i)} \overset{\text{(b)}}{=} \sum_{i=0}^{n-1} w(\rho_i)$$

where Equality (a) follows since $v(x)$ is square free and $d = \mathsf{resultant}(v, f_n)$, and Equality (b) follows since $v(\rho_i) = d/w(\rho_i)$ holds in all the roots of f_n. It is left to show that the constant term of $w(x)$ is $w_0 = n \sum_{i=0}^{n-1} w(\rho_i)$. To show this, we write

$$\sum_{i=0}^{n-1} w(\rho_i) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} w_j \rho_i^j = \sum_{j=0}^{n-1} w_j \sum_{i=0}^{n-1} \rho_i^j \overset{(\star)}{=} \sum_{j=0}^{n-1} w_j \sum_{i=0}^{n-1} (\rho^j)^{2i+1} \qquad (7)$$

where the Equality $(\star)$ holds since the i'th root of f_n is $\rho_i = \rho^{2i+1}$ where ρ is a $2n$-th root of unity. Clearly, the term corresponding to $j = 0$ in Equation (7) is $w_0 \cdot n$, it is left to show that all the other terms are zero. This follows since ρ^j is a $2n$-th root of unity different from ± 1 for all $j = 1, 2, \ldots, n-1$, and summing over all odd powers of such root of unity yields zero.

Step three: recovering the rest of w. We can now use the same technique to recover all the other coefficients of w: Note that since we work modulo $f_n(x) = x^n + 1$, then the coefficient w_i is the free term of the scaled inverse of $x^i \times v \pmod{f_n}$.

In our case we only need to recover the first two coefficients, however, since we are only interested in the case where $w_1/w_0 = w_2/w_1 = \cdots = w_{n-1}/w_{n-2} = -w_0/w_{n-1} \pmod{d}$, where $d = \mathsf{resultant}(v, f_n)$. After recovering w_0, w_1 and

$d = \mathsf{resultant}(v, f_n)$, we therefore compute the ratio $r = w_1/w_0 \bmod d$ and verify that $r^n = -1 \pmod d$. Then we recover as many coefficients of w as we need (via $w_{i+1} = [w_i \cdot r]_d$), until we find one coefficient which is an odd integer, and that coefficient is the secret key.

4.1 The gory details of step one

We denote $U_0(x) \equiv 1$ and $V_0(x) = v(x)$, and for $j = 0, 1, \ldots, \log n$ we denote $n_j = n/2^j$. We proceed in $m = \log n$ steps to compute the polynomials $U_j(x), V_j(x)$ $(j = 1, 2, \ldots, m)$, such that the degrees of U_j, V_j are at most $n_j - 1$, and moreover the polynomial $g_j(z) = \prod_{i=0}^{n_j-1}(V_j(\rho_i^{2^j}) - zU_j(\rho_i^{2^j}))$ has the same first two coefficients as $g(z)$. Namely,

$$g_j(z) \stackrel{\text{def}}{=} \prod_{i=0}^{n_j-1}\Big(V_j(\rho_i^{2^j}) - zU_j(\rho_i^{2^j})\Big) \;=\; g(z) \pmod{z^2}. \tag{8}$$

Equation (8) holds for $j = 0$ by definition. Assume that we computed U_j, V_j for some $j < m$ such that Equation (8) holds, and we show how to compute U_{j+1} and V_{j+1}. From Equation (6) we know that $\big(\rho_{i+n_j/2}\big)^{2^j} = -\rho_i^{2^j}$, so we can express g_j as

$$\begin{aligned} g_j(z) &= \prod_{i=0}^{n_j/2-1}\Big(V_j(\rho_i^{2^j}) - zU_j(\rho_i^{2^j})\Big)\Big(V_j(-\rho_i^{2^j}) - zU_j(-\rho_i^{2^j})\Big) \\ &= \prod_{i=0}^{n_j/2-1}\Big(\underbrace{V_j(\rho_i^{2^j})V_j(-\rho_i^{2^j})}_{=A_j(\rho_i^{2^j})} - z\big(\underbrace{U_j(\rho_i^{2^j})V_j(-\rho_i^{2^j}) + U_j(-\rho_i^{2^j})V_j(\rho_i^{2^j})}_{=B_j(\rho_i^{2^j})}\big)\Big) \pmod{z^2} \end{aligned}$$

Denoting $f_{n_j}(x) \stackrel{\text{def}}{=} x^{n_j} + 1$ and observing that ${\rho_i}^{2^j}$ is a root of f_{n_j} for all i, we next consider the polynomials:

$$A_j(x) \stackrel{\text{def}}{=} V_j(x)V_j(-x) \bmod f_{n_j}(x) \;\; \text{(with coefficients } a_0, \ldots, a_{n_j-1})$$
$$B_j(x) \stackrel{\text{def}}{=} U_j(x)V_j(-x) + U_j(-x)V_j(x) \bmod f_{n_j}(x) \;\; \text{(with coefficients } b_0, \ldots, b_{n_j-1})$$

and observe the following:

- Since ${\rho_i}^{2^j}$ is a root of f_{n_j}, then the reduction modulo f_{n_j} makes no difference when evaluating A_j, B_j on ${\rho_i}^{2^j}$. Namely we have $A_j(\rho_i^{2^j}) = V_j(\rho_i^{2^j})V_j(-\rho_i^{2^j})$ and similarly $B_j(\rho_i^{2^j}) = U_j(\rho_i^{2^j})V_j(-\rho_i^{2^j}) + U_j(-\rho_i^{2^j})V_j(\rho_i^{2^j})$ (for all i).
- The odd coefficients of A_j, B_j are all zero. For A_j this is because it is obtained as $V_j(x)V_j(-x)$ and for B_j this is because it is obtained as $R_j(x) + R_j(-x)$ (with $R_j(x) = U_j(x)V_j(-x)$). The reduction modulo $f_{n_j}(x) = x^{n_j} + 1$ keeps the odd coefficients all zero, because n_j is even.

We therefore set

$$U_{j+1}(x) \stackrel{\text{def}}{=} \sum_{t=0}^{n_j/2-1} b_{2t} \cdot x^t, \quad \text{and} \quad V_{j+1}(x) \stackrel{\text{def}}{=} \sum_{t=0}^{n_j/2-1} a_{2t} \cdot x^t,$$

so the second bullet above implies that $U_{j+1}(x^2) = B_j(x)$ and $V_{j+1}(x^2) = A_j(x)$ for all x. Combined with the first bullet, we have that

$$\begin{aligned} g_{j+1}(z) &\stackrel{\text{def}}{=} \prod_{i=0}^{n_j/2-1} \left(V_{j+1}(\rho_i^{2^{j+1}}) - z \cdot U_{j+1}(\rho_i^{2^{j+1}})\right) \\ &= \prod_{i=0}^{n_j/2-1} \left(A_j(\rho_i^{2^j}) - z \cdot B_j(\rho_i^{2^j})\right) \;=\; g_j(z) \pmod{z^2}. \end{aligned}$$

By the induction hypothesis we also have $g_j(z) = g(z) \pmod{z^2}$, so we get $g_{j+1}(z) = g(z) \pmod{z^2}$, as needed.

5 Encryption

To encrypt a bit $b \in \{0,1\}$ with the public key B (which is implicitly represented by the two integers d, r), we first choose a random $0, \pm 1$ "noise vector" $\boldsymbol{u} \stackrel{\text{def}}{=} \langle u_0, u_1, \ldots, u_{n-1}\rangle$, with each entry chosen as 0 with some probability q and as ± 1 with probability $(1-q)/2$ each. We then set $\boldsymbol{a} \stackrel{\text{def}}{=} 2\boldsymbol{u} + b \cdot \boldsymbol{e}_1 = \langle 2u_0 + b, 2u_1, \ldots, 2u_{n-1}\rangle$, and the ciphertext is the vector

$$\boldsymbol{c} \;=\; \boldsymbol{a} \bmod B \;=\; \boldsymbol{a} - \left(\lceil \boldsymbol{a} \times B^{-1} \rfloor \times B\right) \;=\; \underbrace{\left[\boldsymbol{a} \times B^{-1}\right]}_{[\cdot] \text{ is fractional part}} \times B$$

We now show that also $\boldsymbol{c}$ can be represented implicitly by just one integer. Recall that B (and therefore also B^{-1}) are of a special form

$$B = \begin{bmatrix} d & 0\,0\,0 & 0 \\ -r & 1\,0\,0 & 0 \\ -[r^2]_d & 0\,1\,0 & 0 \\ -[r^3]_d & 0\,0\,1 & 0 \\ & & \ddots \\ -[r^{n-1}]_d & 0\,0\,0 & 1 \end{bmatrix}, \text{ and } B^{-1} = \frac{1}{d} \cdot \begin{bmatrix} 1 & 0\,0\,0 & 0 \\ r & d\,0\,0 & 0 \\ [r^2]_d & 0\,d\,0 & 0 \\ [r^3]_d & 0\,0\,d & 0 \\ & & \ddots \\ [r^{n-1}]_d & 0\,0\,0 & d \end{bmatrix}.$$

Denote $\boldsymbol{a} = \langle a_0, a_1, \ldots, a_{n-1}\rangle$, and also denote by $a(\cdot)$ the integer polynomial $a(x) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} a_i x^i$. Then we have $\boldsymbol{a} \times B^{-1} = \left\langle \frac{s}{d},\ a_1,\ \ldots,\ a_{n-1}\right\rangle$ for some integer s that satisfies $s = a(r) \pmod d$. Hence the fractional part of $\boldsymbol{a} \times B^{-1}$ is $\left[\boldsymbol{a} \times B^{-1}\right] = \left\langle \frac{[a(r)]_d}{d},\ 0,\ \ldots,\ 0\right\rangle$, and the ciphertext vector is $\boldsymbol{c} = \left\langle \frac{[a(r)]_d}{d},\ 0,\ \ldots,\ 0\right\rangle \times B = \langle [a(r)]_d,\ 0,\ \ldots,\ 0\rangle$. Clearly, this vector can

be represented implicitly by the integer $c \stackrel{\text{def}}{=} \left[a(r)\right]_d = \left[b + 2\sum_{i=1}^{n-1} u_i r^i\right]_d$. Hence, to encrypt the bit b, we only need to evaluate the noise-polynomial $u(\cdot)$ at the point r, then multiply by two and add the bit b (everything modulo d). We now describe an efficient procedure for doing that.

5.1 An efficient encryption procedure

The most expensive operation during encryption is evaluating the degree-$(n-1)$ polynomial u at the point r. Polynomial evaluation using Horner's rule takes $n-1$ multiplications, but it is known that for small coefficients we can reduce the number of multiplications to only $O(\sqrt{n})$, see [1, 10]. Moreover, we observe that it is possible to batch this fast evaluation algorithm, and evaluate k such polynomials in time $O(\sqrt{kn})$.

We begin by noting that evaluating many $0, \pm 1$ polynomials at the same point x can be done about as fast as a naive evaluation of a single polynomial. Indeed, once we compute all the powers $(1, x, x^2, \ldots, x^{n-1})$ then we can evaluate each polynomial just by taking a subset-sum of these powers. As addition is much faster than multiplication, the dominant term in the running time will be the computation of the powers of x, which we only need to do once for all the polynomials.

Next, we observe that evaluating a single degree-$(n-1)$ polynomial at a point x can be done quickly given a subroutine that evaluates two degree-$(n/2-1)$ polynomials at the same point x. Namely, given $u(x) = \sum_{i=0}^{n-1} u_i x^i$, we split it into a "bottom half" $u^{\mathsf{bot}}(x) = \sum_{i=0}^{n/2-1} u_i x^i$ and a "top half" $u^{\mathsf{top}}(x) = \sum_{i=0}^{n/2-1} u_{i+\,d/2} x^i$. Evaluating these two smaller polynomials we get $y^{\mathsf{bot}} = u^{\mathsf{bot}}(x)$ and $y^{\mathsf{top}} = u^{\mathsf{top}}(x)$, and then we can compute $y = u(x)$ by setting $y = x^{n/2} y^{\mathsf{top}} + y^{\mathsf{bot}}$. If the subroutine for evaluating the two smaller polynomials also returns the value of $x^{n/2}$, then we need just one more multiplication to get the value of $y = u(x)$.

These two observations suggest a recursive approach to evaluating the $0, \pm 1$ polynomial u of degree $n-1$. Namely, we repeatedly cut the degree in half at the price of doubling the number of polynomials, and once the degree is small enough we use the "trivial implementation" of just computing all the powers of x. Analyzing this approach, let us denote by $M(k, n)$ the number of multiplications that it takes to evaluate k polynomials of degree $(n-1)$. Then we have $M(k, n) \leq \min(n-1,\ M(2k, n/2) + k + 1)$. To see the bound $M(k, n) \leq M(2k, n/2) + k + 1$, note that once we evaluated the top- and bottom-halves of all the k polynomials, we need one multiplication per polynomial to put the two halves together, and one last multiplication to compute x^n (which is needed in the next level of the recursion) from $x^{n/2}$ (which was computed in the previous level). Obviously, making the recursive call takes less multiplications than the "trivial implementation" whenever $n - 1 > (n/2 - 1) + k + 1$. Also, an easy inductive argument shows that the "trivial implementation" is better when

$n-1 < (n/2-1)+k+1$. We thus get the recursive formula

$$M(k,n) \;=\; \begin{cases} M(2k,n/2)+k+1 & \text{when } n/2 > k+1 \\ n-1 & \text{otherwise.} \end{cases}$$

Solving this formula we get $M(k,n) \leq \min(n-1, \sqrt{2kn})$. In particular, the number of multiplications needed for evaluating a single degree-$(n-1)$ polynomial is $M(1,n) \leq \sqrt{2n}$.

We comment that this "more efficient" batch procedure relies on the assumption that we have enough memory to keep all these partially evaluated polynomials at the same time. In our experiments we were only able to use it in dimensions up to $n = 2^{15}$, trying to use it in higher dimension resulted in the process being killed after it ran out of memory. A more sophisticated implementation could take the available amount of memory into account, and stop the recursion earlier to preserve space at the expense of more running time. An alternative approach, of course, is to store partial results to disk. More experiments are needed to determine what approach yields better performance for which parameters.

5.2 The Euclidean norm of fresh ciphertexts

When choosing the noise vector for a new ciphertext, we want to make it as sparse as possible, i.e., increase as much as possible the probability q of choosing each entry as zero. The only limitation is that we need q to be bounded sufficiently below 1 to make it hard to recover the original noise vector from c. There are two types of attacks that we need to consider: lattice-reduction attacks that try to find the closest lattice point to c, and exhaustive-search/birthday attacks that try to guess the coefficients of the original noise vector (and a combination thereof). Pure lattice-reduction attacks should be thwarted by working with lattices with high-enough dimension, so we concentrate here on exhaustive-search attacks.

Roughly, if the noise vector has ℓ bits of entropy, then we expect birthday-type attacks to be able to recover it in $2^{\ell/2}$ time, so we need to ensure that the noise has at least 2λ bits of entropy for security parameter λ. Namely, for dimension n we need to choose q sufficiently smaller than one so that $2^{(1-q)n} \cdot \binom{n}{qn} > 2^{2\lambda}$.

Another "hybrid" attack is to choose a small random subset of the powers of r (e.g., only 200 of them) and "hope" that they include all the noise coefficients. If this holds then we can now search for a small vector in this low-dimension lattice (e.g., dimension 200). For example, if we work in dimension $n = 2048$ and use only 16 nonzero entries for noise, then choosing 200 of the 2048 entries, we have probability of about $(200/2048)^{16} \approx 2^{54}$ of including all of them (hence we can recover the original noise by solving 2^{54} instances of SVP in dimension 200). The same attack will have success probability only $\approx 2^{-80}$ if we use 24 nonzero entries.

For our public challenges we chose a (somewhat aggressive) setting where the number of nonzero entries in the noise vector is between 15 and 20. We note that

increasing the noise will have only moderate effect on the performance numbers of our fully-homomorphic scheme, for example using 30 nonzero entries is likely to increase the size of the key (and the running time) by only about 5-10%.

6 Decryption

The decryption procedure takes the ciphertext c (which implicitly represents the vector $\boldsymbol{c} = \langle c, 0, \ldots, 0\rangle$) and in principle it also has the two matrices V, W. The vector $\boldsymbol{a} = 2\boldsymbol{u} + b \cdot \boldsymbol{e}_1$ that was used during encryption is recovered as $\boldsymbol{a} \leftarrow \boldsymbol{c} \bmod V = \big[\boldsymbol{c} \times W/d\big]$, and then outputs the least significant bit of the first entry of $\boldsymbol{a}$, namely $b := a_0 \bmod 2$.

The reason that this decryption procedure works is that the rows of V (and therefore also of W) are close to being orthogonal to each other, and hence the operator l_∞-norm of W is small. Namely, for any vector $\boldsymbol{x}$, the largest entry in $\boldsymbol{x} \times W$ (in absolute value) is not much larger than the largest entry in $\boldsymbol{x}$ itself. Specifically, the procedure from above succeeds when all the entries of $\boldsymbol{a} \times W$ are smaller than $d/2$ in absolute value. To see that, note that $\boldsymbol{a}$ is the distance between $\boldsymbol{c}$ and some point in the lattice $\mathcal{L}(V)$, namely we can express $\boldsymbol{c}$ as $\boldsymbol{c} = \boldsymbol{y} \times V + \boldsymbol{a}$ for some integer vector $\boldsymbol{y}$. Hence we have

$$\big[\boldsymbol{c} \times W/d\big] \times V = \big[\boldsymbol{y} \times V \times W/d + \boldsymbol{a} \times W/d\big] \stackrel{(\star)}{=} \big[\boldsymbol{a} \times W/d\big] \times V$$

where the equality $(\star)$ follows since $\boldsymbol{y} \times V \times W/d$ is an integer vector. The vector $\big[\boldsymbol{a} \times W/d\big] \times V$ is supposed to be $\boldsymbol{a}$ itself, namely we need $\big[\boldsymbol{a} \times W/d\big] \times V = \boldsymbol{a} = \big(\boldsymbol{a} \times W/d\big) \times V$. But this last condition holds if and only if $[\boldsymbol{a} \times W/d] = (\boldsymbol{a} \times W/d)$, i.e., $\boldsymbol{a} \times W/d$ is equal to its fractional part, which means that every entry in $\boldsymbol{a} \times W/d$ must be less than $1/2$ in absolute value.

6.1 An optimized decryption procedure

We next show that the encrypted bit b can be recovered by a significantly cheaper procedure: Recall that the (implicitly represented) ciphertext vector $\boldsymbol{c}$ is decrypted to the bit b when the distance from $\boldsymbol{c}$ to the nearest vector in the lattice $\mathcal{L}(V)$ is of the form $\boldsymbol{a} = 2\boldsymbol{u} + b\boldsymbol{e_1}$, and moreover all the entries in $\boldsymbol{a} \times W$ are less than $d/2$ in absolute value. As we said above, in this case we have $[\boldsymbol{c} \times W/d] = [\boldsymbol{a} \times W/d] = \boldsymbol{a} \times W/d$, which is equivalent to the condition $[\boldsymbol{c} \times W]_d = [\boldsymbol{a} \times W]_d = \boldsymbol{a} \times W$. Recall now that $\boldsymbol{c} = \langle c, 0, \ldots, 0\rangle$, hence

$$[\boldsymbol{c} \times W]_d = [c \cdot \langle w_0, w_1, \ldots, w_{n-1}\rangle]_d = \langle [cw_0]_d, [cw_1]_d, \ldots, [cw_{n-1}]_d\rangle\,.$$

On the other hand, we have

$$[\boldsymbol{c} \times W]_d = \boldsymbol{a} \times W = 2\boldsymbol{u} \times W + b\boldsymbol{e_1} \times W = 2\boldsymbol{u} \times W + b \cdot \langle w_0, w_1, \ldots, w_{n-1}\rangle\,.$$

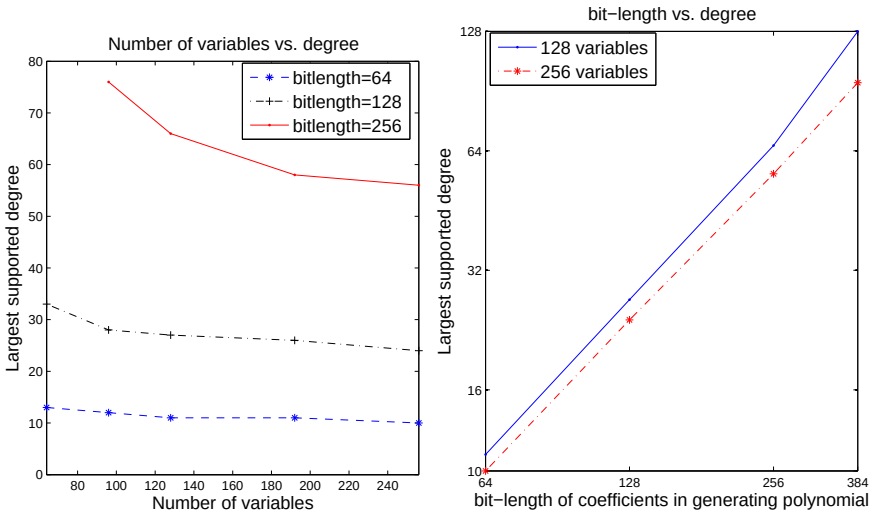
Putting these two equations together, we get that any decryptable ciphertext c must satisfy the relation

$$\langle [cw_0]_d, [cw_1]_d, \ldots, [cw_{n-1}]_d\rangle = b \cdot \langle w_0, w_1, \ldots, w_{n-1}\rangle \pmod 2$$

In other words, for every i we have $[c \cdot w_i]_d = b \cdot w_i \pmod 2$. It is therefore sufficient to keep only one of the w_i's (which must be odd), and then recover the bit b as $b := [c \cdot w_i]_d \mod 2$.

7 How Homomorphic is This Scheme?

We ran some experiments to get a handle on the degree and number of monomials that the somewhat homomorphic scheme can handle, and to help us choose the parameters. In these experiments we generated key pairs for parameters n (dimension) and t (bit-length), and for each key pair we encrypted many bits, evaluated on the ciphertexts many elementary symmetric polynomials of various degrees and number of variables, decrypted the results, and checked whether or not we got back the same polynomials in the plaintext bits.



m =#-of-variables t =bit-length	$m = 64$	$m = 96$	$m = 128$	$m = 192$	$m = 256$
$t = 64$	13	12	11	11	10
$t = 128$	33	28	27	26	24
$t = 256$	64	76	66	58	56
$t = 384$	64	96	128	100	95

Cells contain the largest supported degree for every m, t combination

Fig. 1. Supported degree vs. number of variables and bit-length of the generating polynomial, all tests were run in dimension $n = 128$

More specifically, for each key pair we tested polynomials on 64 to 256 variables. For every fixed number of variables m we ran 12 tests. In each test we

encrypted m bits, evaluated all the elementary symmetric polynomials in these variables (of degree up to m), decrypted the results, and compared them to the results of applying the same polynomials to the plaintext bits. For each setting of m, we recorded the highest degree for which all 12 tests were decrypted to the correct value. We call this the "*largest supported degree*" for those parameters.

In these experiments we used fresh ciphertexts of expected Euclidean length roughly $2 \cdot \sqrt{20} \approx 9$, regardless of the dimension. This was done by choosing each entry of the noise vector $\boldsymbol{u}$ as 0 with probability $1 - \frac{20}{n}$, and as ± 1 with probability $\frac{10}{n}$ each. With that choice, the degree of polynomials that the somewhat-homomorphic scheme could evaluate did not depend on the dimension n: We tested various dimensions from 128 to 2048 with a few settings of t and m, and the largest supported degree was nearly the same in all these dimensions. Thereafter we tested all the other settings only in dimension $n = 128$.

The results are described in Figure 1. As expected, the largest supported degree grows linearly with the bit-length parameter t, and decreases slowly with the number of variables (since more variables means more terms in the polynomial).

These results can be more or less explained by the assumptions that the decryption radius of the secret key is roughly 2^t, and that the noise in an evaluated ciphertext is roughly $c^{\text{degree}} \times \sqrt{\#\text{-of-monomials}}$, where c is close to the Euclidean norm of fresh ciphertexts (i.e., $c \approx 9$). For elementary symmetric polynomials, the number of monomials is exactly $\binom{m}{\text{deg}}$. Hence to handle polynomials of degree deg with m variables, we need to set t large enough so that $2^t \geq c^{\text{deg}} \times \sqrt{\binom{m}{\text{deg}}}$, in order for the noise in the evaluated ciphertexts to still be inside the decryption radius of the secret key.

Trying to fit the data from Figure 1 to this expression, we observe that c is not really a constant, rather it gets slightly smaller when t gets larger. For $t = 64$ we have $c \in [9.14, 11.33]$, for $t = 128$ we have $c \in [7.36, 8.82]$, for $t = 256$ we get $c \in [7.34, 7.92]$, and for $t = 384$ we have $c \in [6.88, 7.45]$. We speculate that this small deviation stems from the fact that the norm of the individual monomials is not exactly c^{deg} but rather has some distribution around that size, and as a result the norm of the sum of all these monomials differs somewhat from $\sqrt{\#\text{-of-monomials}}$ times the expected c^{deg}.

Acknowledgments. We thank Nigel Smart for many excellent comments. We also thank the CRYPTO reviewers for their helpful comments and Tal Rabin, John Gunnels, and Grzegorz Swirszcz for interesting discussions.

References

1. Avanzi, R.M.: Fast evaluation of polynomials with small coefficients modulo an integer. Web document, `http://caccioppoli.mac.rub.de/website/papers/trick.pdf` (2005)
2. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 31–51. Springer (2008)

3. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st ACM Symposium on Theory of Computing – STOC 2009. pp. 169–178. ACM (2009)
4. Gentry, C.: Toward basing fully homomorphic encryption on worst-case hardness. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010. Lecture Notes in Computer Science, vol. 6223, pp. 116–137. Springer (2010)
5. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520 (2010), `http://eprint.iacr.org/`
6. Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reduction problems. In: Jr., B.S.K. (ed.) Advances in Cryptology - CRYPTO 1997. Lecture Notes in Computer Science, vol. 1294, pp. 112–131. Springer (1997)
7. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT'10. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer (2010)
8. Micciancio, D.: Improving lattice based cryptosystems using the hermite normal form. In: CaLC'01. Lecture Notes in Computer Science, vol. 2146, pp. 126–145. Springer (2001)
9. Ogura, N., Yamamoto, G., Kobayashi, T., Uchiyama, S.: An improvement of key generation algorithm for gentry's homomorphic encryption scheme. In: Advances in Information and Computer Security - IWSEC 2010. Lecture Notes in Computer Science, vol. 6434, pp. 70–83. Springer (2010)
10. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing 2(1), 60–66 (1973)
11. Peikert, C., Rosen, A.: Lattices that admit logarithmic worst-case to average-case connection factors. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing – STOC 2007. pp. 478–487. ACM (2007)
12. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation. pp. 169–177. Academic Press (1978)
13. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) Public Key Cryptography - PKC 2010. Lecture Notes in Computer Science, vol. 6056, pp. 420–443. Springer (2010)
14. Stehlé, D., Steinfeld, R.: Faster fully homomorphic encryption. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 377–394. Springer (2010)