Fully Homomorphic Encryption over the Integers with Shorter Public Keys

Jean-Sébastien Coron[1], Avradip Mandal[1],
David Naccache[2], and Mehdi Tibouchi[1,2]

[1] Université du Luxembourg
{jean-sebastien.coron, avradip.mandal}@uni.lu
[2] École normale supérieure
{david.naccache, mehdi.tibouchi}@ens.fr

Abstract. At Eurocrypt 2010 van Dijk *et al.* described a fully homomorphic encryption scheme over the integers. The main appeal of this scheme (compared to Gentry's) is its conceptual simplicity. This simplicity comes at the expense of a public key size in $\tilde{\mathcal{O}}(\lambda^{10})$ which is too large for any practical system. In this paper we reduce the public key size to $\tilde{\mathcal{O}}(\lambda^{7})$ by encrypting with a quadratic form in the public key elements, instead of a linear form. We prove that the scheme remains semantically secure, based on a stronger variant of the approximate-GCD problem, already considered by van Dijk *et al.*

We also describe the first implementation of the resulting fully homomorphic scheme. Borrowing some optimizations from the recent Gentry-Halevi implementation of Gentry's scheme, we obtain roughly the same level of efficiency. This shows that fully homomorphic encryption can be implemented using simple arithmetic operations.

1 Introduction

Fully Homomorphic Encryption. An encryption scheme is homomorphic if it supports operations on encrypted data. For example RSA is multiplicatively homomorphic since $c_1 = m_1^e \mod N$ and $c_2 = m_2^e \mod N$ yield the encryption of $m_1 \cdot m_2$ without using the private key.

Similarly, Paillier cryptosystem [12] is additively homomorphic because from $c_1 = g^{m_1} r^N \mod N^2$ and $c_2 = g^{m_2} s^N \mod N^2$ one can compute the encryption of $m_1 + m_2$.

In a breakthrough work Gentry described in 2009 the first encryption scheme that supports both addition and multiplication on ciphertexts, *i.e.* a fully homomorphic encryption scheme [5]. The construction proceeds by successive steps: First Gentry describes a "somewhat homomorphic" scheme that supports a limited number of additions and multiplications on ciphertexts. This is because every ciphertext has a noise component and any homomorphic operation applied to ciphertexts increases the noise in the resulting ciphertext. Once this noise reaches a certain threshold the resulting ciphertext does not decrypt correctly anymore; this limits the degree of the polynomial that can be applied to ciphertexts.

Secondly Gentry shows how to "squash" the decryption procedure so that it can be expressed as a low degree polynomial in the bits of the ciphertext and the secret key (equivalently a circuit of small depth). Then the breakthrough idea consists in evaluating this decryption polynomial not on the bits of the ciphertext and the secret key (as in regular decryption), but homomorphically on the *encryption* of those bits. Then instead of recovering the bit plaintext, one gets an encryption of this bit plaintext, *i.e.* yet another ciphertext for the same plaintext; see Figure 1 for an illustration. Now if the degree of the decryption polynomial is small enough, the resulting noise in this new ciphertext can be smaller than in the original ciphertext; this is called the "ciphertext refresh" procedure. Given two refreshed ciphertexts one can apply again the homomorphic operation (either addition or multiplication), which was not necessarily possible on the original ciphertexts because of the noise threshold. Using this "ciphertext refresh" procedure the number of permissible homomorphic operations becomes unlimited and we get a fully homomorphic encryption scheme.

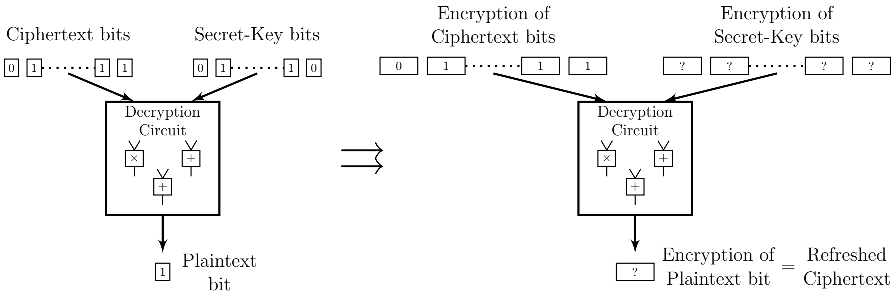

Fig. 1. The decryption circuit applied on the ciphertext bits and secret key bits (left), and the ciphertext refresh procedure with the decryption circuit applied homomorphically on the encryption of those bits (right).

The prerequisite for the "ciphertext refresh" procedure is that the degree of the polynomial that can be evaluated on ciphertexts exceeds the degree of the decryption polynomial (times two, since one must allow for a subsequent addition or multiplication of refreshed ciphertexts); this is called the "bootstrappability" condition. Once the scheme becomes bootstrappable it can be converted into a fully homomorphic encryption scheme by providing the encryption of the secret key bits inside the public key.

Based on Gentry's approach, two different fully homomorphic schemes are known: Gentry's scheme [5] based on ideal lattices and a scheme by van Dijk, Gentry, Halevi and Vaikuntanathan (DGHV) over the integers, that appeared at Eurocrypt 2010 [4].

Gentry's scheme and its implementations. Gentry described in [5] a somewhat homomorphic encryption scheme that is similar to GGH [7, 14] over ideal lattices. To reduce the degree of the decryption polynomial, Gentry introduced the following transformation [5]: instead of using the original secret key, the decryption procedure uses a very sparse subset of values that adds up to the secret key; the full set of values is made part of the public key. To apply the new decryption procedure the original ciphertext must first be "expanded" using the full set of public values. This expanded ciphertext can then be decrypted with a low-degree polynomial in the bits of the new secret key (which are the characteristic vector of the sparse subset sum); this is called the "squashed decryption" procedure.

At PKC 2010 Smart and Vercauteren [16] made the first attempt to implement Gentry's scheme using a variant based on principal ideal lattices and requiring that the determinant of the lattice be a prime number. However the authors of [16] could not obtain a bootstrappable scheme because that would have required a lattice dimension of at least $n = 2^{27}$, whereas due to the prime determinant requirement they could not generate keys for dimensions $n > 2048$.

Gentry and Halevi described in [6] the first implementation of Gentry's scheme. The authors follow the same direction as Smart and Vercauteren, but for key generation they eliminate the requirement that the determinant is a prime. Additionally they present many clever optimizations. Four concrete parameter settings are provided, from a "toy" setting in dimension 512, to "small", "medium" and "large" settings of dimensions 2048, 8192 and 32768, respectively. For the "large" setting public key size is 2.3 Gigabytes. The authors of [6] report that for an optimized implementation on a high-end workstation, key generation takes 2.2 hours, encryption takes 3 minutes, and ciphertext refresh takes 30 minutes.

The DGHV fully homomorphic scheme over the integers. At Eurocrypt 2010, van Dijk, Gentry, Halevi and Vaikuntanathan described a fully homomorphic encryption scheme over the integers [4]. As in Gentry's scheme the authors first describe a somewhat homomorphic scheme supporting a limited number of additions and multiplications over encrypted bits. Then they apply Gentry's "squash decryption" technique to get a bootstrappable scheme and then Gentry's "ciphertext refresh" procedure (see Fig. 1) to get a fully homomorphic scheme.

The main appeal of the scheme (compared to the original Gentry's scheme) is its conceptual simplicity; namely all operations are done over the integers instead of ideal lattices. However the public-key was in $\tilde{\mathcal{O}}(\lambda^{10})$ which is too large for any practical system.

Our Contributions. In this paper we show how to reduce the public key size of the somewhat homomorphic scheme from $\mathcal{O}(\lambda^{10})$ down to $\mathcal{O}(\lambda^{7})$. The idea consists in storing only a smaller subset of the public key and then generating the full public key on the fly by combining the elements in the small subset multiplicatively; we describe the new scheme in Section 3. In Section 4 we show that the new scheme is still semantically secure, but under a stronger variant of the approximate GCD assumption.

Our second contribution is to describe an implementation of the fully homomorphic DGHV scheme under our variant, using some of the optimizations from [6]. We use the refined analysis from [17] of the sparse subset sum problem; however we do not use the probabilistic decryption circuit from [17] because as in [6] the error probability is too high for our set of parameters. The main difficulty is to determine a secure set of concrete parameters; our approach is to implement the known attacks, measure their running time and extrapolate for large parameters; we can then fix the concrete parameters according to the desired level of security.

We obtain similar performances as the Gentry-Halevi implementation of Gentry's scheme [6]. More precisely we use four security levels inspired by the levels from [6] (though they may not be directly comparable due to different notions of "security bits"): "toy", "small", "medium" and "large", corresponding to 42, 52, 62 and 72 bits of security respectively. For "large" parameters, encryption and recryption take 3 minutes and 14 minutes respectively, with a public key size of 800 MBytes. Decryption is always close to instantaneous. This shows that fully homomorphic encryption can be implemented with a simple scheme.

2 The DGHV Scheme over the Integers.

In this section we first recall the somewhat homomorphic encryption scheme published by van Dijk, Gentry, Halevi and Vaikuntanathan at Eurocrypt 2010 [4]. The scheme is based on a set of public integers: $x_i = p \cdot q_i + r_i$, $0 \le i \le \tau$, where the integer p is secret.

Notation. We use the same notation as in [4]. For a real number x, we denote by $\lceil x \rceil$, $\lfloor x \rfloor$ and $\lceil x \rfloor$ the rounding of x up, down, or to the nearest integer. For integers z, p we denote the reduction of z modulo p by $[z]_p$ with $-p/2 < [z]_p \le p/2$. We also denote $[z]_p$ by $z \mod p$. We write $f(\lambda) = \tilde{\mathcal{O}}(g(\lambda))$ if $f(\lambda) = \mathcal{O}(g(\lambda) \log^k g(\lambda))$ for some $k \in \mathbb{N}$.

The scheme parameters. Given the security parameter λ, the following parameters are used:

- γ is the bit-length of the x_i's.
- η is the bit-length of secret key p.
- ρ is the bit-length of the noise r_i.
- τ is the number of x_i's in the public key.
- ρ' is a secondary noise parameter used for encryption.

For a specific η-bit odd integer p, we use the following distribution over γ-bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) = \left\{ \text{ Choose } q \leftarrow \mathbb{Z} \cap [0, 2^\gamma/p), r \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho) : \text{ Output } x = q \cdot p + r \right\}$$

$\mathsf{KeyGen}(1^\lambda)$. Generate a random odd integer p of size η bits. For $0 \leq i \leq \tau$ sample $x_i \leftarrow \mathcal{D}_{\gamma,\rho}(p)$. Relabel so that x_0 is the largest. Restart unless x_0 is odd and $[x_0]_p$ is even. Let $pk = (x_0, x_1, \ldots x_\tau)$ and $sk = p$.

$\mathsf{Encrypt}(pk, m \in \{0, 1\})$. Choose a random subset $S \subseteq \{1, 2, \ldots, \tau\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output the ciphertext:

$$c = \left[m + 2r + 2 \sum_{i \in S} x_i \right]_{x_0}$$

$\mathsf{Evaluate}(pk, C, c_1, \ldots, c_t)$: given the circuit C with t input bits, and t ciphertexts c_i, apply the addition and multiplication gates of C to the ciphertexts, performing all the additions and multiplications over the integers, and return the resulting integer.

$\mathsf{Decrypt}(sk, c)$. Output $m \leftarrow (c \bmod p) \bmod 2$. Note that since $c \bmod p = c - p \cdot \lfloor c/p \rceil$ and p is odd, one can compute instead: $m \leftarrow [c]_2 \oplus [\lfloor c/p \rceil]_2$.

This completes the description of the scheme. It is shown in [4] that the scheme is a somewhat homomorphic scheme and that it is semantically secure under the approximate-GCD assumption.

Definition 2.1 (Approximate GCD). *The (ρ, η, γ)-approximate-GCD problem is: For a random η-bit odd integer p, given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$, output p.*

Note that after one Mult operation $c \leftarrow c_1 \cdot c_2$ the ciphertext size doubles since there is no modular reduction involved. To reduce the ciphertext size after one Mult two techniques are described in [4]. The second and simpler technique consists in generating x_0 without noise, that is $x_0 = q_0 \cdot p$, and then reducing the ciphertext modulo x_0. The scheme is still semantically secure under the (stronger) approximate-GCD assumption with error-free x_0. While this problem seems easier to solve, as the adversary is given an exact multiple of p, no better attack is known against it than on the unmodified problem.

We recall in the full version of this paper [3] the constraints on the scheme parameters. To satisfy theses constraints the following parameter set is suggested in [4]: $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = \tilde{\mathcal{O}}(\lambda^2)$, $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ and $\tau = \gamma + \lambda$. The public key size is then $\tilde{\mathcal{O}}(\lambda^{10})$. In practice the size of the x_i's should be at least $\gamma = 2^{23}$ bits to prevent lattice attacks. The public key size is then at least 2^{46} bits, which is too large for any practical system.

3 Our Variant of the DGHV Scheme

3.1 Description

Our technique consists in working with integers x'_{ij} of the form $x'_{i,j} = x_{i,0} \cdot x_{j,1} \bmod x_0$ for $1 \leq i, j \leq \beta$ where β is a new parameter. Then only 2β integers $x_{i,b}$

need to be stored in the public key in order to generate the $\tau = \beta^2$ integers x'_{ij} used for encryption. In other words we encrypt using a quadratic form in the public key elements instead of a linear form, which enables to reduce the public key size from τ down to roughly $2\sqrt{\tau}$ integers of γ bits.

Our technique requires to use an error-free x_0, that is $x_0 = q_0 \cdot p$, since otherwise the error would grow too large. Additionally for encryption we consider a linear combination of the $x'_{i,j}$ with coefficients in $[0, 2^\alpha)$ instead of bits; this enables to further reduce the public key size.

$\mathsf{KeyGen}(1^\lambda)$. Generate a random prime $p \in \cap[2^{\eta-1}, 2^\eta)$. Let $x_0 = q_0 \cdot p$ where q_0 is a random square free 2^λ-rough[1] integer in $[0, 2^\gamma/p)$. Generate integers $x_{i,b}$ for $1 \leq i \leq \beta$ and $b \in \{0, 1\}$:

$$x_{i,b} = p \cdot q_{i,b} + r_{i,b}, \quad 1 \leq i \leq \beta,\ 0 \leq b \leq 1 \tag{1}$$

where $q_{i,b}$ are random integers in $[0, q_0)$ and $r_{i,b}$ are integers in $(-2^\rho, 2^\rho)$. Let $sk = p$ and $pk = (x_0, x_{1,0}, x_{1,1}, \ldots x_{\beta,0}, x_{\beta,1})$.

$\mathsf{Encrypt}(pk, m \in \{0, 1\})$. Generate a random vector $\boldsymbol{b} = (b_{i,j})$ of size $\tau = \beta^2$ and with components in $[0, 2^\alpha)$. Generate a random integer r in $(-2^{\rho'}, 2^{\rho'})$. Output the ciphertext:

$$c = m + 2r + 2 \sum_{1 \leq i,j \leq \beta} b_{i,j} \cdot x_{i,0} \cdot x_{j,1} \mod x_0 \tag{2}$$

$\mathsf{Evaluate}$ and $\mathsf{Decrypt}$: same as in the original scheme, except that ciphertexts are reduced modulo x_0 after addition and multiplication.

3.2 Constraints on the Parameters

The first three constraints are the same as in the original DGHV scheme:

- $\rho = \omega(\log \lambda)$ to avoid brute force attack on the noise (see Section 6.1).
- $\eta \geq (2\rho + \alpha) \cdot \Theta(\lambda \log^2 \lambda)$ in order to support homomorphic operations for evaluating the "squashed decryption circuit" (see Section 5).
- $\gamma = \omega(\eta^2 \cdot \log \lambda)$ in order to thwart lattice-based attacks (see Section 6).
- $\alpha \cdot \beta^2 \geq \gamma + \omega(\log \lambda)$ for the reduction to approximate GCD (see Section 4).
- $\rho' = 2\rho + \alpha + \omega(\log \lambda)$ for the secondary noise parameter (see Section 4).

To satisfy these conditions we can still take $\rho = \lambda$, $\eta = \tilde{\mathcal{O}}(\lambda^2)$ and $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ as in the original scheme, and we can take $\alpha = \lambda$, $\beta = \tilde{\mathcal{O}}(\lambda^2)$ and $\rho' = 4\lambda$. The main difference is that instead of having $\tau = \tilde{\mathcal{O}}(\lambda^5)$ integers x_i's, we now have only $2\beta = \tilde{\mathcal{O}}(\lambda^2)$ integers x_i. Hence the public key size becomes $\tilde{\mathcal{O}}(\lambda^7)$ instead of $\tilde{\mathcal{O}}(\lambda^{10})$. In Section 7.5 we describe concrete parameters in order to resist all known attacks.

[1] An integer is said to be a-rough when it does not contain prime factors smaller than a. Note that for $a > 2$ such integer must be odd.

Remark 3.1. It is possible to generate q_0 as a uniformly random square free 2^λ-rough integer of suitable size in probabilistic polynomial time: it suffices to generate a uniformly random number with known factorization [1] and try again if it has small or repeated factors. However, this makes key generation rather unpractical. Alternatively, one can choose q_0 as the product of $(\gamma - \eta)/\lambda^2$ random primes, each of size λ^2 bits.[2] This is faster, but the security of the scheme then depends on a slightly more convoluted, though no less plausible, computational assumption, to account for the different key distribution.

3.3 Correctness

We refer to the full version of this paper [3] for the definition of correct homomorphic scheme with respect to a given circuit or circuit set. As in [4, 5] we define a *permitted circuit* as one where for any $i \geq 1$ and any set of integer inputs all less than $\tau^i \cdot 2^{i(\rho'+2)}$ in absolute value, the generalized circuit's output has absolute value at most $2^{i(\eta-3-n)}$ with $n = \lceil \log_2(\lambda+1) \rceil$; we let $\mathcal{C}_\mathcal{E}$ be the set of permitted circuits. As in [4], we have (see proof in the full version of this paper [3]):

Lemma 3.1. *The scheme from above is correct for $\mathcal{C}_\mathcal{E}$.*

Remark 3.2. Since "fresh" ciphertexts output by Encrypt have noise at most $\tau \cdot 2^{\rho'+2}$, the ciphertext output by Evaluate applied to a permitted circuit has noise at most $2^{\eta-3-n} < p/(4(\lambda+1))$. A bound of $p/2$ would suffice to ensure correct decryption, but this stronger bound will be useful to prove the correctness of the bootstrappable version of this scheme later on.

Remark 3.3. The definition of a permitted circuit doesn't seem to give an easy criterion to determine whether a given computation is permitted. However, it is easy to give a sufficient condition on a multivariate polynomial f for the associated arithmetic circuit C to be permitted. If f is of degree d and if the sum of the absolute values of its coefficients is denoted by $\|f\|_1$, then $C \in \mathcal{C}_\mathcal{E}$ provided that:

$$d \leq \frac{\eta - 3 - n - \log \|f\|_1}{\rho' + 2 + 2\log\beta}$$

Following [4], we refer to such polynomials f as *permitted polynomials*, and denote the set of these polynomials by $\mathcal{P}_\mathcal{E}$.

4 Security of our Variant

4.1 Overview

In this section we show that our variant is still semantically secure, but under the (stronger) error-free approximate GCD assumption. Our security proof follows

[2] The reason we choose λ^2-bit factors rather than λ is because factorization algorithms like ECM have a complexity subexponential in the size of factors, and can thus be used to extract λ-bit prime factors efficiently. In the implementation, to thwart this attack, it is safe to generate q_0 as a product of, say, 1000-bit primes.

the same strategy as in [4]: show that an adversary breaking the scheme's semantic security can be converted into a LSB predictor for $z \bmod p$, where z is an integer such that $z \bmod p$ is small; this in turns enables to recover p in the approximate-GCD problem.

For this one must show that given $c \leftarrow \mathsf{Encrypt}(pk, m)$, the distribution of $c' = [c+z]_{x_0}$ is essentially the same as $\mathsf{Encrypt}(pk, m')$ with $m' = m \oplus [z \bmod p]_2$. In [4] this is done by showing that the distribution of $\lfloor c/p \rceil = \sum_{i=1}^{\tau} b_i \cdot q_i$ where $\boldsymbol{b} \leftarrow \{0,1\}^\tau$ is statistically close to uniform in $\mathbb{Z}_{q_0}$. For this [4] applies the leftover hash lemma on the hash function family $h(\boldsymbol{b}) = \sum_{i=1}^{\tau} b_i \cdot q_i \mod q_0$ parametrized by the q_i's, which is clearly pairwise independent.

Similarly to prove the security of our variant we must apply the leftover hash lemma on the hash function family $h' : [0, 2^\alpha)^{\beta^2} \to \mathbb{Z}_{q_0}$ where:

$$h'(\boldsymbol{b}) = \sum_{1 \le i,j \le \beta} b_{i,j} \cdot q_{i,0} \cdot q_{j,1} \mod q_0$$

The main difficulty is to show that h' is (almost) pairwise independent; as shown below this requires to study the zeroes of the corresponding quadratic form. We note that our result might be of independent interest since it enables to construct a universal hash function with a small memory footprint.

4.2 Leftover Hash Lemma

A family $\mathcal{H}$ of hash functions $h : X \to Y$ is pairwise independent if for all $x \neq x'$ it holds that $\Pr_h[h(x) = h(x')] = 1/|Y|$. Since h' is not exactly pairwise independent we introduce a slightly more general definition:[3]

Definition 4.1. *A family $\mathcal{H}$ of hash functions $h : X \to Y$ is ε-pairwise independent if*

$$\sum_{x \neq x'} \left(\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] - \frac{1}{|Y|} \right) \le |X|^2 \cdot \frac{\varepsilon}{|Y|}$$

The following lemma is a straightforward generalization of the usual leftover hash lemma. We provide the proof in the full version of this paper [3].

Lemma 4.1 (Leftover hash lemma). *Let $\mathcal{H}$ be a family of ε-pairwise independent hash functions. Suppose that $h \leftarrow \mathcal{H}$ and $x \leftarrow X$ are chosen uniformly and independently. Then $(h, h(x))$ is $(\frac{1}{2}\sqrt{|Y|/|X| + \varepsilon})$-uniform over $\mathcal{H} \times Y$.*

[3] Note that this is quite different from "ε-almost universal hash function families" in the sense of Wegman and Carter [19]. We need the collision probability $\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')]$ to be at most $(1+\varepsilon)/|Y|$ on average, with negligible ε; $2/|Y|$ is not good enough.

4.3 Proof of Pairwise Independence

Let q be an integer. Let $\mathcal{H}$ be a hash function family from $\{0, \ldots, 2^\alpha - 1\}^{\beta\times\beta}$ to $\mathbb{Z}_q$. The members $h \in \mathcal{H}$ are associated to elements $q_{i,0}, q_{i,1}$ of $\mathbb{Z}_q$ for $1 \leq i \leq \beta$. For $\boldsymbol{b} \in \{0, \ldots, 2^\alpha - 1\}^{\beta\times\beta}$, we let:

$$h(\boldsymbol{b}) = \sum_{1\leq i,j\leq\beta} b_{ij} q_{i,0} q_{j,1} \mod q$$

Lemma 4.2. *For an odd prime integer q, the hash function family $\mathcal{H}$ is ε-pairwise independent, with:*

$$\varepsilon = \frac{1}{q} + \frac{\beta^2}{2^{\alpha\beta^2 - 2(\alpha+1)\beta}}$$

Proof. For each choice of $\boldsymbol{b} \neq \boldsymbol{b'}$, the probability $\Pr_{h\leftarrow\mathcal{H}}[h(\boldsymbol{b}) = h(\boldsymbol{b'})]$ can be expressed in terms of the number of zeros of a certain hyperbolic quadratic form in $\mathbb{Z}_q^{2\beta}$. More precisely let $A = (a_{ij})$ be the $\beta\times\beta$ matrix in $\mathbf{M}_\beta(\mathbb{Z}_q)$ given by $a_{ij} = b_{ij} - b'_{ij}$. We have:

$$\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] = \frac{1}{q^{2\beta}} \#\left\{(u_1, \ldots, u_\beta, v_1, \ldots, v_\beta) \in \mathbb{Z}_q^{2\beta} \ : \sum_{1\leq i,j\leq\beta} a_{ij} u_i v_j = 0\right\}$$

Now the quadratic form $Q = \sum_{1\leq i,j\leq\beta} a_{ij} u_i v_j$ has the matrix $\frac{1}{2}\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$, which is clearly conjugate to $\frac{1}{2}\begin{pmatrix} 0 & J \\ J & 0 \end{pmatrix}$ where J is the canonical row echelon form of A. It follows that Q is the orthogonal sum of r hyperbolic planes, with r the rank of A. Hence, its number of zeros is well-known (see e.g. [9, Theorem 6.32] for the non-degenerate case, from which the general case follows immediately):

$$\#\left\{(u_1, \ldots, u_\beta, v_1, \ldots, v_\beta) \in \mathbb{Z}_q^{2\beta} : \sum_{1\leq i,j\leq\beta} a_{ij} u_i v_j = 0\right\} = q^{2\beta-1} + q^{2\beta-r} - q^{2\beta-r-1}$$

In particular, we get:

$$\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] - \frac{1}{q} \leq \frac{1}{q^r}$$

This estimate is quite sufficient for our purposes, except in the case when $r = 1$. Therefore, we need to bound the number of pairs $(\boldsymbol{b}, \boldsymbol{b'})$ such that the corresponding matrix A is of rank 1. Noting that A has all its entries in $-2^\alpha + 1, \ldots, 2^\alpha - 1$, it is enough to bound the cardinality of the set U_α of matrices of rank 1 in $M_\beta(\mathbb{Z}_q)$ with entries in that interval.

To do so, note that a matrix of rank 1 with a nonzero upper-left entry is entirely determined by its first line and its first column. If the entries are in $\{-2^\alpha + 1, \ldots, 2^\alpha - 1\}$, this leaves $2^{\alpha+1} - 2$ choices for the upper-left entries and $(2^{\alpha+1} - 1)^{2\beta-2}$ choices for the remainder of the first line and the first column. Hence, there are less than $2^{2(\alpha+1)(\beta-1)}$ matrices in U_α with a nonzero upper-left entry (and usually much fewer, since not all first lines and first columns need to give rise to matrices with all their entries in the proper interval). The same argument can be applied to any other nonzero entry (i, j), leading to the coarse bound:

$$|U_\alpha| < \beta^2 \cdot 2^{2(\alpha+1)\beta}$$

Now, the number of pairs $(\boldsymbol{b}, \boldsymbol{b'})$ such that the corresponding matrix A is of rank 1 is at most $|X| \cdot |U_\alpha|$, since for any choice of $\boldsymbol{b}$, there are at most $|U_\alpha|$ possible values of $\boldsymbol{b'}$ such that A is in U_α. We can thus bound the value δ defined by:

$$\delta = \frac{|Y|}{|X|^2} \sum_{\boldsymbol{b} \neq \boldsymbol{b'}} \left(\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] - \frac{1}{|Y|} \right)$$

as required. Indeed:

$$\begin{aligned} \delta = \frac{q}{|X|^2} \sum_{\boldsymbol{b} \neq \boldsymbol{b'}} \left(\Pr_h[h(\boldsymbol{b}) = h(\boldsymbol{b'})] - \frac{1}{q} \right) &\leq \frac{q}{|X|^2} \left(\sum_{\substack{\boldsymbol{b} \neq \boldsymbol{b'} \\ A \notin U_\alpha}} \frac{1}{q^2} + \sum_{\substack{\boldsymbol{b} \neq \boldsymbol{b'} \\ A \in U_\alpha}} \frac{1}{q} \right) \\ \leq \frac{q}{|X|^2} \left(\frac{|X|^2}{q^2} + \frac{|X| \cdot |U_\alpha|}{q} \right) \leq \frac{1}{q} + \frac{|U_\alpha|}{|X|} &\leq \frac{1}{q} + \frac{\beta^2}{2^{\alpha\beta^2 - 2(\alpha+1)\beta}} \end{aligned}$$

which concludes the proof. □

Corollary 4.1. *When q is a product of distinct primes greater than 2^α, the hash function family $\mathcal{H}$ is ε-pairwise independent, with:*

$$\varepsilon = \frac{\log q}{\log p} \left(\frac{e}{p} + \frac{\beta^2 \cdot 2^{(\log q)/(\log p)}}{2^{\alpha\beta^2 - 2(\alpha+1)\beta}} \right) \quad \textit{where } p \textit{ is the smallest prime factor of } q.$$

Proof. The proof is largely similar to the previous one. See the full version of this paper [3] for details.

4.4 Semantic Security

We are now ready to show that our variant is semantically secure under the (stronger) error-free approximate GCD assumption. The proof follows the same strategy as [4]; we refer to the full version of this paper [3] for the details. For two specific integers p and q_0, we define the modified distribution:

$$\mathcal{D}'_\rho(p, q_0) = \left\{ \textit{ Choose } q \leftarrow [0, q_0), r \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho) : \textit{ Output } x = q \cdot p + r \right\}$$

Definition 4.2 (Error-free approximate GCD). *The (ρ, η, γ)-error-free-approximate-GCD problem is: For a random η-bit prime integer p, given $x_0 = q_0 \cdot p$ where q_0 is a random square free 2^λ-rough integer in $[0, 2^\gamma/p)$, and polynomially many samples from $\mathcal{D}'_\rho(p, q_0)$, output p.*

Theorem 4.1. *Let $\mathcal{A}$ be an attacker with advantage ε against our variant encryption scheme with parameters $(\rho, \rho', \eta, \gamma, \tau = \beta^2)$ polynomial in the security parameter λ. There exists an algorithm $\mathcal{B}$ for solving the (ρ, η, γ)-error-free-approximate-GCD problem that succeeds with probability at least $\varepsilon/2$. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$, λ and $1/\varepsilon$.*

5 Making the Scheme Fully Homomorphic

5.1 The Squashed Scheme

Gentry's transformation to "squash the decryption" consists in adding to the public key some extra information about the secret key and use this extra information to "post process" the ciphertext. Then the post-processed ciphertext can be decrypted by a decryption polynomial of small degree. This requires to introduce an additional complexity assumption, namely the sparse subset-sum assumption.

We follow the description of [4]. Three more parameters κ, θ and Θ are added. Concretely, one uses $\theta = \lambda$, $\kappa = \gamma + 2 + \lceil \log_2(\theta + 1) \rceil$, and $\Theta = \tilde{\mathcal{O}}(\lambda^3)$.[4] One adds to the public key a set $\boldsymbol{y} = \{y_1, \ldots, y_\Theta\}$ of rational numbers in $[0, 2)$ with κ bits of precision, such that there is a sparse subset $S \subset \{1, \ldots, \Theta\}$ of size θ with $\sum_{i \in S} y_i \simeq 1/p \mod 2$. The expanded ciphertext is computed using the y_i's. The secret key sk is replaced by the indicator vector of the subset S.

However adding Θ elements y_i each of size κ bits would give a public key of size $\Theta \cdot \kappa = \tilde{\mathcal{O}}(\lambda^8)$, instead of $\tilde{\mathcal{O}}(\lambda^7)$ in our variant. Therefore instead of storing the y_i's in the public key as in [4], we generate the y_i's using a pseudo-random generator[5] $f(\mathsf{se})$. Then only the seed se and y_1 need to be stored in the public key, and the other y_i's can be recovered during ciphertext expansion by applying $f(\mathsf{se})$ again. We obtain the following squashed scheme:

KeyGen. Generate $sk^* = p$ and pk^* as before. Set $x_p \leftarrow \lfloor 2^\kappa / p \rceil$, choose at random a Θ-bit vector $\boldsymbol{s} = (s_1, \ldots, s_\Theta)$ with Hamming weight θ with $s_1 = 1$, and let $S = \{i : s_i = 1\}$.

Initialize a system-wide pseudo-random number generator f with a random seed se, and use $f(\mathsf{se})$ to generate integers $u_i \in [0, 2^{\kappa+1})$ for $2 \le i \le \Theta$. Then, set u_1 such that $\sum_{i \in S} u_i = x_p \mod 2^{\kappa+1}$. Set $y_i = u_i / 2^\kappa$ and $\boldsymbol{y} = \{y_1, \ldots, y_\Theta\}$. Hence each y_i is a positive number smaller than two, with κ bits of precision after the binary point. Also, $[\sum_{i \in S} y_i]_2 = (1/p) - \Delta_p$ for some $|\Delta_p| < 2^{-\kappa}$.

Output the secret key $sk = \boldsymbol{s}$ and public key $pk = (pk^*, \boldsymbol{y})$.

Encrypt and Evaluate. Generate a ciphertext c^* as before. Then for $i \in \{1, \ldots, \Theta\}$ set $z_i \leftarrow [c^* \cdot y_i]_2$, keeping only $n = \lceil \log_2(\theta + 1) \rceil$ bits of precision after the binary point for each z_i. Output both c^* and $\boldsymbol{z} = (z_1, \ldots, z_\Theta)$.

Decrypt: Output $m \leftarrow [c^* - \lfloor \sum_i s_i z_i \rceil]_2$.

This completes the description of the scheme. Note that as in [6] we use $n = \lceil \log_2(\theta + 1) \rceil$ bits of precision, instead of $n = \lceil \log_2 \theta \rceil + 3$ in the original scheme. This enables to reduce the degree of the decryption polynomial. In

[4] We use $\Theta = \tilde{\mathcal{O}}(\lambda^3)$ instead of $\Theta = \omega(\kappa \cdot \log \lambda)$ in [4] from a better analysis of the hardness of the SSSP problem (see Section 6.3).

[5] Note that f doesn't really need to be a cryptographically strong PRNG: all that is needed is that the sparse subset-sum problem remains hard when the subset is generated by f. Heuristically, this is a mild requirement. In our implementation, we use random numbers produced by the PRNG from the `glibc`.

practice we will use $n = 4$. Note that for encryption we don't need to store all the y_i's in memory again; we can generate them one by one from the PRNG to compute $z_i \leftarrow [c^* \cdot y_i]_2$ with n bits of precision.

The proof of the following lemma is similar to the one in [4] (see the full version of this paper [3]), but we can handle a smaller precision n, as in [6], because in our scheme, ciphertext size does not grow in homomorphic operations.

Lemma 5.1. *The modified scheme is correct for the set $C(\mathcal{P}_{\mathcal{E}})$ of circuits that compute permitted polynomials.*

5.2 Bootstrapping

As in [4], one obtains that the scheme is bootstrappable. From Gentry's theorem we obtain homomorphic encryption schemes for circuits of any depth.

Theorem 5.1. *Let $\mathcal{E}$ be the scheme above, and let $D_{\mathcal{E}}$ be the set of augmented (squashed) decryption circuits. Then, $D_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$.*

Proof. The proof is as in [4]. We provide a slightly different analysis. We consider the decryption equation:

$$m \leftarrow c^* - \left\lfloor \sum_{i=i}^{\Theta} s_i \cdot z_i \right\rceil \mod 2$$

where s_i are the secret key bits and z_i are rational numbers in $[0, 2)$ with n bits of precision after the binary point (therefore $n+1$ bits in total). We must express the decryption equation as a low degree polynomial in the bits s_i and the bits in z_i, *i.e.* a permitted polynomial.

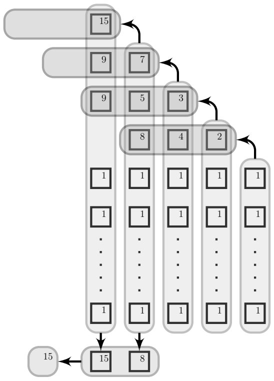

Fig. 2. Grade-school addition for Θ or $\theta = 15$ numbers of $n = 4$ bits of precision after the binary point. The numbers indicate the degree of each bit as a binary polynomial in the input bits.

For this one uses a simple grade-school addition of the numbers $a_i = s_i \cdot z_i$. As illustrated in Fig. 2 the bits of the a_i's are arranged in Θ rows and $n+1$ columns (one column before the binary point and n columns after). To see how this grade-school addition can be performed efficiently, first recall the following result from [4, §6.2].

Lemma 5.2. *Let $\boldsymbol{b} = (b_1, b_2, \ldots, b_\Theta)$ be any binary vector, and denote its Hamming weight by W. Write the binary digits of W as $W = \overline{W_k \cdots W_1 W_0}^2$. Then the j-th bit W_j of W can be expressed as a binary polynomial of degree exactly 2^j in the b_i's, namely the 2^j-th elementary symmetric polynomial:*

$$W_j = \sum_{\substack{I \subset \{1,\ldots,\Theta\} \\ |I| = 2^j}} \prod_{i \in I} b_i$$

Moreover, the bits $W_0, W_1, \ldots W_j$ can be simultaneously computed by an arithmetic circuit of size $2^j \cdot \Theta$.

That the W_j's are given by elementary symmetric polynomials is classical (see e.g. [2, Lemma 4]). Thus, to compute them, it suffices to find the top 2^j coefficients of the polynomial $(X - b_1)(X - b_2) \cdots (X - b_\Theta)$, which can be done iteratively with at most $2^j \cdot \Theta$ operations. We recall in the full version of this paper [3] the dynamic programming algorithm from [4].

Then, the procedure to compute

$$Q = \left\lfloor \sum_{k=1}^{\Theta} a_k \right\rceil$$

is as follows. We number the columns containing the a_k's from left to right as 0 (before the binary point), -1, -2, …, $-n$.

As usual, grade-school addition starts from the rightmost column (column $-n$). Adding all Θ bits from that column produces a bit of result and a certain number of bits of carry. Since we are only interested in the $n+1$ least significant bits of the sum, we only need to keep track of the result and the first n carry bits: this amounts to computing the rightmost bits $W_0^{(-n)}, W_1^{(-n)}, \ldots, W_n^{(-n)}$ of the Hamming weight $W^{(-n)}$ of column $-n$, which can be done with at most $2^n \cdot \Theta$ multiplications according to the previous lemma.

Now, push carry bit $W_1^{(-n)}$ to column $-n+1$, carry bit $W_2^{(-n)}$ to column $-n+2$ and so on. We can then continue the grade-school addition process from column $-n+1$, where we only need to compute the result and $n-1$ carry bits, namely the bits $W_j^{(-n+1)}$ of the Hamming weight $W^{(-n+1)}$ of the column, including the possible carry bit from column $-n$. This amounts to at most $2^{n-1} \cdot (\Theta + 1)$ multiplications. When this is done, push the carry bits $W_1^{(-n+1)}$, $\ldots W_{n-1}^{(-n+1)}$ to columns $-n+2$, $-n+3$, …, 0 respectively, move to the next column and continue as before. This is illustrated in Figure 2 for $n = 4$.

As shown in the full version of this paper [3], this can be done in $\mathcal{O}(\Theta \cdot \theta)$ multiplications, and the decryption polynomial is a polynomial f of the ciphertext bits and the secret key satisfying $d = \deg f = 2^{n+1}$ and $\|f\|_1 \leq 2\Theta$. In view of Remark 3.3, f is a permitted polynomial as long as $d \leq (\eta - 4 - n - \log_2 \Theta)/(\rho' + 2 + 2\log_2 \beta) \approx \eta/\rho'$ which is satisfied by choosing η according to the constraint in Section 3.2. □

6 Attacks

In this section we recall the known attacks. For each attack we provide an asymptotic analysis (as in [4]) and we also run the attacks in practice in order to derive concrete parameters for our implementation. We use four security levels inspired by the levels from [6]: "toy", "small", "medium" and "large", corresponding to 42, 52, 62 and 72 bits of security respectively. For security parameter λ we wish to ensure that the best attack requires at least 2^λ clock cycles on a standard PC.

Note that we use the SAGE [13] interface to the `fplll` lattice reduction package [15] which is to our knowledge the fastest publicly available. However any progress in LLL implementations will require an increase of our security parameters.

6.1 Brute Force Attack on the Noise

The easiest attack is the brute force attack on the noise in the public key. Given $x_0 = q_0 \cdot p$ and $x_1 = q_1 \cdot p + r_1$ with $|r_1| < 2^\rho$, one can guess r_1 and compute $\gcd(x_0, x_1 - r_1)$ to recover p. The state of the art algorithm for computing GCD's is the Stehlé-Zimmermann algorithm [18] with time complexity $\tilde{\mathcal{O}}(\gamma)$ for integers of γ bits. The attack complexity is then $2^\rho \cdot \tilde{\mathcal{O}}(\gamma)$. Therefore the attack is thwarted if $\rho = \omega(\log \lambda)$.

A better attack [11] consists in computing $p = \gcd(x_0, \prod_{i=-2^\rho}^{2^\rho}(x_1 - i)\ [x_0])$. Using fast multiplication the asymptotic complexity is also $2^\rho \cdot \tilde{\mathcal{O}}(\gamma)$. Experimentally this later attack is roughly 5 times faster. See the full version of this paper [3] for concrete parameters.

6.2 Approximate-GCD Attack on the Public Key

We do not consider Coppersmith's attack since as shown in [4] it does not apply for the range of parameters. We consider the attack based on Nguyen and Stern's orthogonal lattice [10] (see Section B.1 in [4]). One considers the first $t \leq \tau$ integers $x_i = p \cdot q_i + r_i$ and $x_0 = p \cdot q_0$. The attack builds the lattice L of row vectors orthogonal to $\boldsymbol{x} = (x_1, \ldots, x_t)$ modulo x_0 (see the full version of this paper [3] for more details). One must find a vector $\boldsymbol{u} \in L$ such that $\|u\|_\infty \leq 2^{\eta-\rho}$. From Minkowski's bound there exists a nonzero lattice vector of norm about $\sqrt{t} \cdot \det(L)^{1/t} \simeq 2^{\gamma/t}$, which gives the condition $t > \gamma/\eta$. However when the lattice dimension t is large, lattice reduction algorithms will not recover such a short vector but only an approximation.

As in [4] we use the following "rule of thumb" conjecture about lattice algorithms performance: there exists a constant μ such that for any k and any dimension t, one cannot find a $\mu^{t/k}$ approximation of the shortest vector in time smaller than 2^k. Since we must find a vector $\boldsymbol{u}$ such that $\|\boldsymbol{u}\|_\infty \leq 2^{\eta-\rho}$, we need better than a $2^{\eta-\rho}$ approximation of the shortest vector. To get a 2^η approximation (which is not quite enough to recover $\boldsymbol{u}$), from $t > \gamma/\eta$ the time required is then at least 2^k where $k = (\log_2 \mu)\gamma/\eta^2$. We recover the asymptotic condition $\gamma = \eta^2 \cdot \omega(\log \lambda)$. To obtain concrete parameters we have run some experiments with the LLL and BKZ-20 lattice reduction algorithms; see the full version of this paper [3].

6.3 Lattice Attack on the Sparse Subset-sum Problem

We use the refined analysis from [17] of the sparse subset sum problem. The attacker must solve the equation $\sum_{i=1}^{\Theta} s_i \cdot u_i = x_p \mod 2^\kappa$ where $\boldsymbol{s} = (s_1, \ldots, s_\Theta)$ is of small Hamming weight θ.

As shown in the full version of this paper [3] the lattice attack leads to a lattice L of determinant $\det L \simeq 2^{\Theta+\kappa} \simeq 2^{\Theta+\gamma}$. The lattice has a short vector of norm about $\sqrt{\Theta}$. From Minkowsky's bound we can expect that the norm of the second shortest vector is $\simeq (\det L)^{1/\Theta} \simeq 2^{\gamma/\Theta}$. Therefore to find the shortest vector we need better than a $2^{\gamma/\Theta}$ approximation. From the lattice "Rule of Thumb" conjecture with the previous notations the time required is then at least 2^k with $k = (\log_2 \mu)\Theta^2/\gamma$. Asymptotically the condition is therefore $\Theta^2 = \gamma \cdot \omega(\log \lambda)$. Therefore with $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ we can take $\Theta = \tilde{\mathcal{O}}(\lambda^3)$. We refer to the full version of this paper [3] for concrete parameters; we also consider a birthday-like attack on the sparse subset-sum problem.

7 Implementation of the Fully Homomorphic Scheme

7.1 Recryption

Now that decryption can be expressed as an arithmetic circuit of low depth, it is possible to achieve bootstrapping, i.e. to publicly evaluate the decryption circuit homomorphically on a ciphertext, which produces another ciphertext for the same message, but with possibly less noise (a "recryption"). This process, which is Gentry's key idea [5] for achieving fully homomorphic encryption, is illustrated in Figure 1. The procedure that evaluates the decryption circuit homomorphically, called Recrypt, takes as input encryptions of the ciphertext bits, and encryptions of the secret key bits.

In the case of the DGHV scheme or of our variant, 0 and 1 are valid encryptions of themselves, so the ciphertext bits can be passed as is to the decryption circuit. However, encryptions of the secret key bits should also be made available publicly, so the key generation from §5.1 should be modified to include encryptions σ_i of the s_i's into the public key $pk = (pk^*, \boldsymbol{y}, \boldsymbol{\sigma})$. Then the Recrypt procedure is simply obtained by applying the decryption circuit to the ciphertext bits and the encrypted secret key bits.

Note that such ciphertexts σ_i are normally generated using the $x_{i,b}$'s from the public key, leading to σ_i's with noise of size ρ'. However since we are at key generation phase we can do better and let $\sigma_i = s_i + 2r'_i + 2p \cdot q'_i \mod x_0$ for $1 \leq i \leq \Theta$, for random integers q'_i modulo q_0 and random integers r'_i in $(-2^\rho, 2^\rho)$. The resulting ciphertexts σ_i have the same distribution as regular ciphertexts but with noise ρ instead of ρ'. Since $\rho < \rho'$ this enables to reduce the size η of p required to achieve bootstrappability.

For the refreshed ciphertext to decrypt correctly, its noise must be small enough, and in fact small enough that a multiplication operation between refreshed ciphertexts still decrypts correctly. The ciphertext bits are noise-free encryptions of themselves and the encrypted secret key bits contain ρ bits of noise, so one must have $d \cdot \rho < \eta/2$, where d is the degree of the decryption circuit discussed in the previous section (or in fact, only half that degree, because only the degree with respect to the secret key bits matters; the contribution in the ciphertext bits z_i can be ignored as they are used directly and without noise).

7.2 Optimization of the Decryption Circuit

We use the optimization from [4] which consists in representing the secret key $\boldsymbol{s}$ in θ boxes of $B = \Theta/\theta$ bits each, such that each box has a single 1-bit in it. This enables to obtain a grade-school addition algorithm that requires $\mathcal{O}(\theta^2)$ multiplications of bits instead of $\mathcal{O}(\Theta \cdot \theta)$. We refer to the full version of this paper [3] for the details. Note in particular that it results from the analysis that the degree of the decryption polynomial in the secret key bits is exactly θ. See also Fig. 2 for an illustration of the grade-school addition algorithm with $n = 4$.

7.3 Compression of Encrypted Secret Key Bits

The modification of the public key described previously, to accommodate for the Recrypt procedure, has the problem of increasing public key size significantly. Namely the vector $\boldsymbol{\sigma}$ in the enlarged public key consists of $\Theta = \tilde{\mathcal{O}}(\lambda^3)$ ciphertexts, each of size $\gamma = \tilde{\mathcal{O}}(\lambda^5)$, so we obtain a public key size of $\tilde{\mathcal{O}}(\lambda^8)$, instead of $\tilde{\mathcal{O}}(\lambda^7)$ in the basic scheme.

To alleviate this problem, an additional compression trick is described in [6]. Instead of generating the secret key as a single bit vector $\boldsymbol{s} = (s_1, \ldots, s_\Theta)$, one uses two bit vectors $\boldsymbol{s}^{(0)}$ and $\boldsymbol{s}^{(1)}$ of length $\sqrt{\Theta}$, and $\boldsymbol{s}$ is then recovered on the fly during decryption with $s_{i,j} = s_i^{(0)} \cdot s_j^{(1)}$. See the full version of this paper [3] for the details. This brings down the size of the encrypted secret key bits to about $\sqrt{\Theta} \cdot \gamma = \tilde{\mathcal{O}}(\lambda^{6.5})$. Note on the other hand that this increases the noise in $\boldsymbol{\sigma}$ by a factor of 2 since the $\sigma_{i,j}$ are obtained as products of two ciphertexts; this implies that to keep bootstrappability the size η of p must be doubled.

7.4 Smaller Dimension for Knapsack Encryption

From the previous section the size of the public key in the full scheme is now about $(\beta + \sqrt{\Theta}) \cdot \gamma$ bits. The conditions from Section 3.2 imply that we must have $\beta = \tilde{\mathcal{O}}(\lambda^2)$ to apply the leftover hash lemma. Since $\sqrt{\Theta} = \tilde{\mathcal{O}}(\lambda^{1.5})$ we have that β is the bottleneck. Therefore in practice we would like to use a smaller β, for which the leftover hash lemma would not apply but no attack would work.

This implies that we must consider a lattice attack against the knapsack sum in the encryption algorithm. The analysis is the same as in Section 6.3, with $\tau = \beta^2$ instead of Θ. This gives the asymptotic condition $\tau^2 = \gamma \cdot \omega(\log \lambda)$ which for $\alpha < \tau$ is weaker than the condition $\alpha \cdot \tau \geq \gamma + \omega(\log \lambda)$ necessary for the reduction to the approximate GCD problem. Under this condition we can take $\tau = \tilde{\mathcal{O}}(\gamma^3)$ instead of $\tau = \tilde{\mathcal{O}}(\gamma^4)$ and therefore $\beta = \tilde{\mathcal{O}}(\gamma^{1.5})$ instead of $\beta = \tilde{\mathcal{O}}(\gamma^2)$. The public key size is then $(\beta + \sqrt{\Theta}) \cdot \gamma = \tilde{\mathcal{O}}(\lambda^{6.5})$ instead of $\tilde{\mathcal{O}}(\lambda^7)$.

7.5 Concrete Parameters

From the analysis of the known attacks in the previous section we are now ready to derive the concrete parameters for the four levels of security. For all four levels we take $\theta = 15$. In this case the degree of the decryption polynomial is $2\theta = 30$ when using the degree-2 compression of the encryption of the secret key bits. Since we must allow for an additional multiplication after Recrypt, the total degree is $d = 4 \cdot \theta = 60$. To allow for some margin we take $\eta = (d + 8)\rho = 68 \cdot \rho$. We obtain the parameters given in Table 1.

Parameters	λ	ρ	η	γ	β	Θ
Toy	42	16	1088	$1.6 \cdot 10^5$	12	144
Small	52	24	1632	$0.86 \cdot 10^6$	23	533
Medium	62	32	2176	$4.2 \cdot 10^6$	44	1972
Large	72	39	2652	$19 \cdot 10^6$	88	7897

Parameters	KeyGen	Encrypt	Expand	Decrypt	Recrypt	pk size
Toy	4.38 s	0.05 s	0.03 s	0.01 s	1.92 s	0.95 MB
Small	36 s	0.79 s	0.46 s	0.01 s	10.5 s	9.6 MB
Medium	5 min 9 s	10 s	8.1 s	0.02 s	1 min 20 s	89 MB
Large	43 min	2 min 57 s	3 min 55 s	0.05 s	14 min 33 s	802 MB

Table 1. Concrete parameters and corresponding timings, as measured using our implementation in Sage 4.5.3 [13] and GMP 4.3.2 [8], on a single core of a desktop computer with an Intel Core2 Duo E8500 CPU at 3.12 GHz. The public key is roughly $2(\beta + \sqrt{\Theta} + 1)\gamma$ bit long. Note that almost all the CPU time of key generation is spent in primality tests, to generate a rough q_0.

Acknowledgments

We would like to thank Phong Q. Nguyen, Nigel P. Smart and the CRYPTO referees for helpful comments. The work described in this paper has been supported in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

References

1. E. Bach, *How to generate factored random numbers*. SIAM J. Comput., vol. 17, 1988, pp. 179–193.
2. J. Boyar, R. Peralta and D. Pochuev, *On the multiplicative complexity of boolean functions over the basis* $(\wedge, \oplus, 1)$. Theor. Comput. Sci., vol. 235(1), 2000, pp. 43–57.
3. J.S. Coron, A. Mandal, D. Naccache and M. Tibouchi, *fully Homomorphic Encryption over the Integers with Shorter Public Keys*. Full version of this paper, available at `http://eprint.iacr.org`.
4. M. van Dijk, C. Gentry, S. Halevi and V. Vaikuntanathan, *Fully homomorphic encryption over the integers*. In H. Gilbert (Ed.), *EUROCRYPT 2010*, LNCS, vol. 6110, Springer, 2010, pp. 24–43.
5. C. Gentry, *A fully homomorphic encryption scheme*. Ph.D. thesis, Stanford University, 2009. Available at `http://crypto.stanford.edu/craig`.
6. C. Gentry and S. Halevi, *Implementing Gentry's fully homomorphic encryption scheme*. In K. Paterson (Ed.), *EUROCRYPT 2011*, LNCS, Springer, 2011. To appear.
7. O. Goldreich, S. Goldwasser and S. Halevi, *Public-key cryptosystems from lattice reduction problems*. In B. S. Kaliski (Ed.), *CRYPTO '97*, LNCS, vol. 1294, Springer, 1997, pp. 112–131.
8. T. Grandlung et al., *The GNU Multiple Precision arithmetic library* (Version 4.3.2), 2010, `http://gmplib.org`.
9. R. Lidl and H. Niederreiter, *Finite fields*. Encyclopedia of mathematics and its applications, vol. 20, Addison-Wesley, 1983.
10. P. Q. Nguyen and J. Stern, *The two faces of lattices in cryptology*. In J. H. Silverman (Ed.), *CaLC 2001*, LNCS, vol. 2146, Springer, pp. 146–180
11. P. Q. Nguyen, personal communication.
12. P. Paillier, *Public-key cryptosystems based on composite degree residuosity classes*. In J. Stern (Ed.), *EUROCRYPT '99*, LNCS, vol. 1592, Spring, 1999, pp. 223–238.
13. W.A. Stein et al., *Sage Mathematics Software* (Version 4.5.3), The Sage Development Team, 2010, `http://www.sagemath.org`.
14. D. Micciancio, *Improving lattice based cryptosystems using the Hermite normal form*. In J. H. Silverman (Ed.), *CaLC 2001*, LNCS, vol. 2146, Springer, pp. 126–145
15. X. Pujol, D. Stehlé et al., `fplll` *lattice reduction library*, `http://perso.ens-lyon.fr/xavier.pujol/fplll/`.
16. N. P. Smart and F. Vercauteren, *Fully homomorphic encryption with relatively small key and ciphertext sizes*. In P. Q. Nguyen and D. Pointcheval (Eds.), *PKC 2010*, LNCS, vol. 6056, Springer, 2010, pp. 420–443.
17. D. Stehlé and R. Steinfeld, *Faster fully homomorphic encryption*. In M. Abe (Ed.), *ASIACRYPT 2010*, LNCS, vol. 6477, Springer, 2010, pp. 377–394.
18. D. Stehlé and P. Zimmermann, *A binary recursive GCD algorithm*. In D. A. Buell, *ANTS-VI*, LNCS, vol. 3076, Springer, 2004, pp. 411–425.
19. M.N. Wegman and J.L. Carter, *New hash functions and their use in authentication and set equality*, Journal of Computer and System Sciences, vol. 22(3), 1981, pp. 265-279.